ENABLING INTRUSION DETECTION IN IPSEC PROTECTED IPV6 NETWORKS THROUGH SECRET-KEY SHARING

THESIS

Patrick J. Sweeney, Captain, USAF

AFIT/GCE/ENG/05-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCE/ENG/05-06

ENABLING INTRUSION DETECTION IN IPSEC PROTECTED IPV6 NETWORKS THROUGH SECRET-KEY SHARING

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Patrick J. Sweeney, BS

Captain, USAF

March 2005

AFIT/GCE/ENG/05-06

ENABLING INTRUSION DETECTION IN IPSEC PROTECTED IPV6 NETWORKS THROUGH SECRET-KEY SHARING

Patrick J. Sweeney, BS

Captain, USAF

Approved:

/signed/ 7 Mar 2005

Dr. Richard A. Raines Date
Thesis Advisor

/signed/ 7 Mar 2005

Dr. Michael A. Temple Date
Committee Member

/signed/ 4 Mar 2005

Dr. Rusty O. Baldwin Date
Committee Member

/signed/ 4 Mar 2005

Dr. Barry E. Mullins Date
Committee Member

Acknowledgments

I would like to express my appreciation to my thesis advisor, Dr. Richard Raines, for his constant support and insightful comments, even after I changed topics away from his specialty. Thank you for being open minded with my ideas, and shaping them into a useful research effort. To Dr. Rusty Baldwin, thank you for helping me figure out how to accomplish my goals, and for the guidance on statistics and performance analysis. To Dr. Temple, credit is given where it’s due—the number was 12. Thanks for making me get my data representation in order. For Dr. Mullins, good luck with this group. Overall, thanks for making this thesis as painless as possible.

Thanks to Capt Dave Chaboya, for helping find all the mistakes in my code, and keeping me motivated by always staying one step ahead of me in the thesis process… if only I can get this to the Kinko’s first, I win!

Finally, thank you to my wife, who put up with all the late homework nights, whining, and nerd-talk without (much) complaint.

Patrick J. Sweeney

Table of Contents

Page

List of Figures

List of Tables

AFIT/GCE/ENG/05-06

Abstract

As the Internet Protocol version 6 (IPv6) implementation becomes more widespread, the IP Security (IPSec) features embedded into the next-generation protocol will become more accessible than ever. Though the network-layer encryption provided by IPSec is a boon to data security, its use renders standard network intrusion detection systems (NIDS) useless. The problem of performing intrusion detection on encrypted traffic has been addressed by differing means with each technique requiring one or more static secret keys to be shared with the NIDS beforehand. The problem with this approach is static keying is much less secure than dynamic key generation through the Internet Key Exchange (IKE) protocol.

This research creates and evaluates a secret-key sharing framework which allows both the added security of dynamic IPSec key generation through IKE, *and* intrusion detection capability for a NIDS on the network. Analysis shows that network traffic related to secret-key sharing with the proposed framework can account for up to 58.6% of total traffic in the worst case scenario, though workloads which are arguably more average decrease that traffic to 10-15%. Additionally, actions associated with IKE and secret-key sharing increase CPU utilization on the NIDS up to 20.7%. Results show, at least in limited implementations, a secret-key sharing framework provides robust coverage and is a viable intrusion detection option.

ENABLING INTRUSION DETECTION IN IPSEC PROTECTED IPV6 NETWORKS THROUGH SECRET-KEY SHARING

1 Introduction

1.1 Background

Network security is a topic of much interest. Incidents involving malicious code such as the Blaster and Code Red worms demonstrate skilled hackers are capable of causing havoc on systems and networks. This, coupled with the fact that Microsoft published 45 security bulletins in 2004, shows there are plenty of vulnerabilities to be exploited in the most prevalent operating systems on the Internet [Mic04]. Many tools, from firewalls to virus scanners, have been developed to aid in network security. An increasingly common method for protecting a network is an Intrusion Detection System (IDS). IDSs come in a variety of types, most notably host-based IDS (HIDS) and network-based IDS (NIDS). HIDS are installed on individual network hosts, as the name suggests, and traditionally examine log files and system registries on the host to determine if an intrusion attempt is underway or has taken place. NIDS, on the other hand, monitor network traffic by examining packets on the network for signs of attack, or attack signatures. Misuse-based NIDS perform pattern matching on Ethernet packets, whereas Anomaly-based NIDS try to determine what "normal" network activity is, and alert operators of any deviations from that model. The former is currently more prevalent.

Although a “defense in depth” approach is a sound security principle, if a choice must be made between a HIDS or NIDS, often a NIDS is selected. This one appliance or software suite protects an entire subnet and is generally easier to manage than numerous HIDSs. A widely acknowledged shortcoming of a NIDS, however, is it cannot examine encrypted traffic [Tri03, Shi02, Fed04, Tay02]. Without the ability to decrypt traffic, the NIDS cannot perform pattern matching for attack signatures. While there are ways to work with encrypted traffic for specialized circumstances, such as Secure Socket Layers and HTTPS traffic [Mca04], those forms of data protection don’t provide as broad a coverage as IPSec which operates at the network layer.

The problem is exacerbated by the advent of Internet Protocol version 6 (IPv6) [Tri03]. This next-generation IP protocol is currently in the implementation phase in many Asian countries, and will replace IPv4 globally in the near future. IPv6 is touted as being more secure, as it natively supports the Authentication Header (AH) and Encapsulating Security Payload (ESP) header which are available to IPv4 by using IPSec [War03]. ESP provides network layer encryption which can be applied to all IP traffic traveling to or from a host. This provides a great deal of security benefit for legitimate traffic, but it has the side effect of allowing malicious traffic to be encrypted as well, denying a NIDS the ability of detecting the attack.

This research creates a secret-key sharing framework enabling a host on an IPv6 network with ESP encryption to decrypt network traffic and send it to a NIDS engine for attack detection. The performance of the system is analyzed across a range of encryption types and traffic loads.

1.2 Problem Definition

1.2.1 Goals and Hypothesis The goals of this research are straightforward. Since NIDSs are currently limited to non-encrypted environments, the overarching purpose is to find a way around that limitation. The first specific goal is to create and implement a secret-key sharing method which allows decryption of network traffic and passing of the same to an intrusion detection engine in the clear. The ability of the system to detect individual attacks is evaluated through empirical testing.

The second goal is to examine traffic patterns generated by providing workloads which vary in the percentage of encrypted traffic and attack payload size. Response to the workloads is studied in scenarios with different network configurations and different keying methods.

The third and final goal compares the performance of the physical IDS host in an IPSec encrypted environment utilizing static keys versus dynamic keys with varying key expiration times. More explicitly, the goal determines how well the secret-key sharing framework enables attack detection on the encrypted traffic by examining the CPU utilization of the computer hosting the IDS. The intrusion detection engine itself acts independent of the framework (although on the same physical host), and is presented clear text data either directly or after decryption by the framework. The likely result is that detection will be hampered by a significant increase in computation required to decrypt the traffic. The comparisons, however, will provide some insight into whether such IDS are feasible in high throughput networks.

Overall, the expectation is a minimal-overhead framework can be developed for secret-key sharing, enabling attack detection on encrypted traffic. The hypothesis is the decryption bottleneck, even in a subnet of limited size, occupies a high enough percentage of CPU cycles on the IDS host as to make the impact of secret-key sharing negligible, lending credence to the idea that a dynamic keying with a secret-key sharing framework is a viable option to static keying.

1.2.2 Approach To accomplish the goals of the research, the secret-key sharing framework is developed. This framework transfers keys from a host to the IDS interface whenever a new key is generated. In addition, it receives keys on the IDS interface, decodes and decrypts Ethernet packets, and passes clear text data to the intrusion detection engine. This is accomplished through modification of an open-source Internet Key Exchange (IKE) implementation, as well as generation of some custom software for the decryption interface.

With the secret-key sharing framework in place, the second and third goals are addressed by subjecting the framework to various factor levels and observing the responses. When all experiments are complete, trends are examined and compared to the hypothesis.

1.3 Summary

This thesis contains four additional chapters. Chapter 2 discusses background information pertinent to the experiments, and the current state of research in the areas of IPv6 and intrusion detection for encrypted traffic. Chapter 3 describes the experimental methodology, and Chapter 4 presents the data and analysis resultant from the experiments. Finally, Chapter 5 presents a the conclusions of the research and explores ideas for future work.

2 Background/Literature Review

2.1 Introduction

This chapter presents an overview of the problems facing network security in light of the implementation of IPv6, and explores concepts important to developing a secret-key sharing framework and Intrusion Detection System (IDS) for an IPSec enabled IPv6 network. Section 2 describes details about IPv6, and how it differs from IPv4. Section 3 gives a more detailed look at the Authentication Header (AH) and Encapsulating Security Payload (ESP) header built into IPv6. Section 4 provides information about the Security Associations which support the AH and ESP headers, as well as the Security Association Databases (SAD) and Security Policy Database (SPD). Section 5 looks at problems that widespread use of the AH/ESP headers pose to Intrusion Detection Systems (IDS), and Section 6 examines the issues that need to be addressed in order to overcome these problems. Finally Section 7 looks at relevant research in the area.

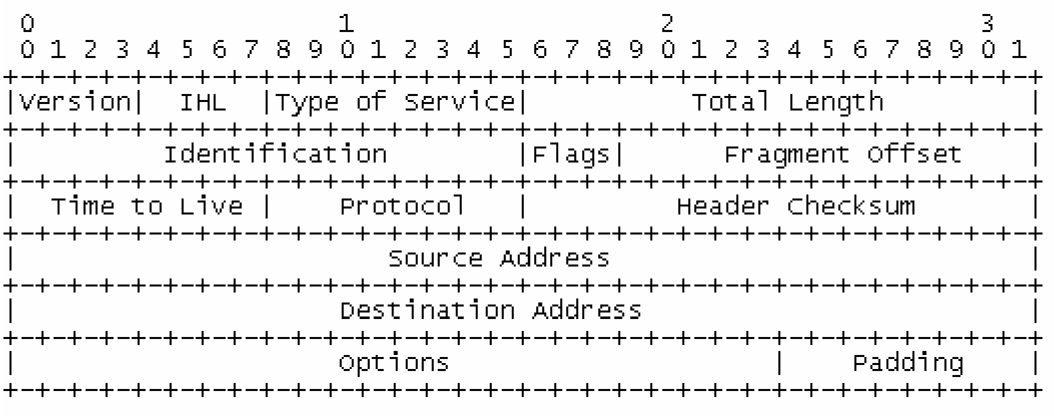
2.2 Internet Protocol Version 6 (IPv6) Basics

IPv4 is an aging protocol. Perhaps the biggest motivation for the evolution to IPv6 is the depletion of the limited address space provided by IPv4. An IPv4 address is comprised of 32 bits that represent a network address. No matter how the available address space is divided among different entities, it can only provide approximately 4 billion IP addresses. When IP was first implemented, Internet activity was dominated by educational institutions, research centers, and other professional organizations. At that time, the 4 billion addresses must have seemed more than sufficient. However, the success of commercial forays on the Internet coupled with the amazing growth over the last 20 years resulted in the address space reaching its limit.

This is especially true in countries other than the US. The US has a fairly high percentage of IP addresses. For example US-based Level 3 Communications owns 3 Class A networks, which gives them a slightly larger address space than all of Asia [Sch04]. For this reason, some countries such as Japan and Korea have mandated implementation of IPv6 by 2005 [Emi02]. IPv6 will, first and foremost, alleviate the address space problem. Rather than the 32-bit address provided by IPv4, IPv6 has a 128-bit address, or approximately 2^128 = 3.40 * 10^{38} addresses.

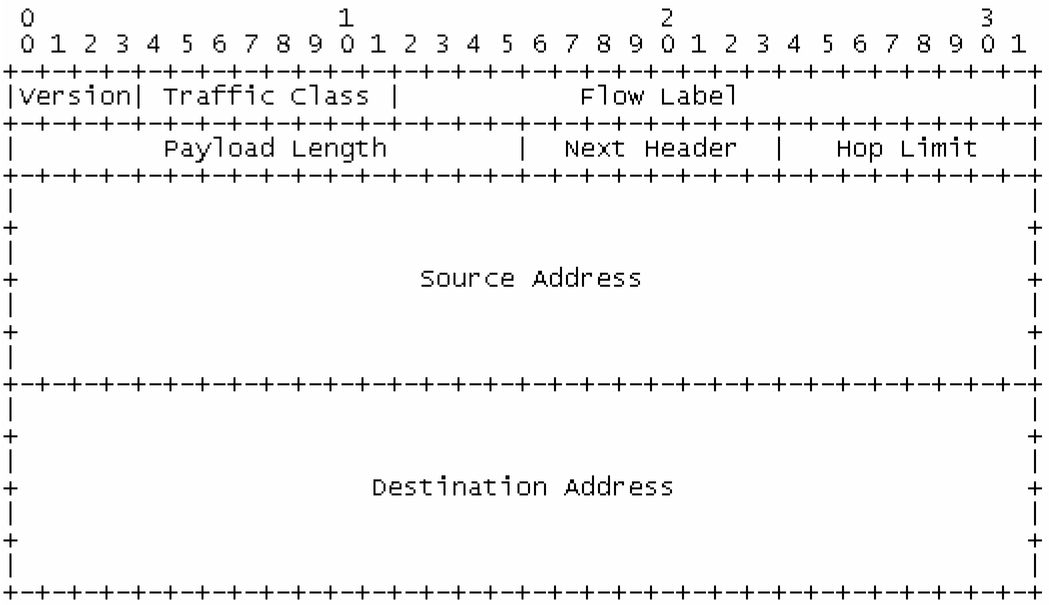
Transition to IPv6 in the US is likely to be slower than some foreign countries, since the US has enough IPv4 addresses to meet near-term needs. Techniques have been developed which prolonged IPv4's success despite the problems. For instance, Network Address Translation (NAT) allows private subnetworks to be connected to the Internet via one public IP address. Classless Inter-Domain Routing (CIDR) provides a more efficient way to allocate IP addresses to organizations, rather than the allocation of only class A, B, and C networks. However, the eventual adoption of IPv6 seems inevitable. This is good given that, among other improvements, IPv6 has built-in authentication and security mechanisms discussed in Section 2.3.

IPv6 improves on IPv4 by standardizing more of the IPv4 "options" and more efficiently using these functions. Additionally, the IPv6 header is more space-efficient than the IPv4 header, as shown in Figures 2.1 and 2.2. The *IHL, type of service*, *identification, flags*, *checksum,* and *options and padding* fields were dropped for IPv6. Even with an address four times longer, the header is only twice as large. The header also provides greater flexibility. For instance, packet priority can be set via the Traffic Class field so real-time applications like streaming video and audio can be given higher priority.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.1: IPv4 Header



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.2: IPv6 Header

The following describes the header fields of the IPv6 datagram [Jav04]:

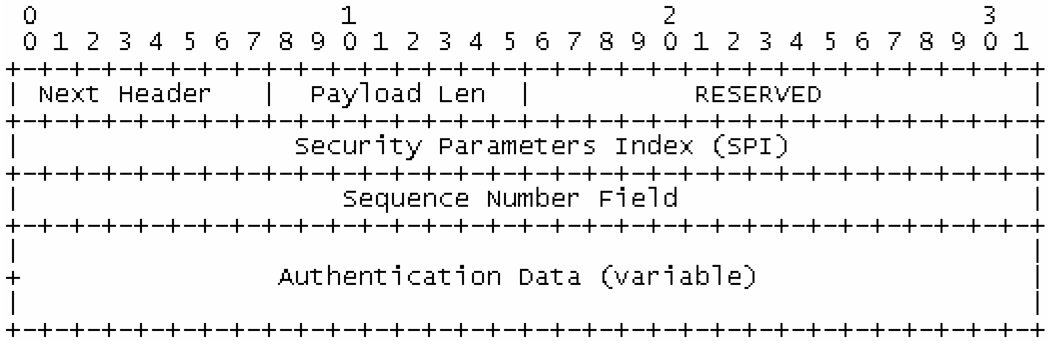
- *Version* -- Internet Protocol Version number (IPv6 is 6).
- *Traffic Class* -- Traffic class field enables a source to identify the desired delivery priority of the packets. Priority values are divided into ranges: traffic where the source provides congestion control and non-congestion control traffic.
- *Flow label* -- Flow label is used by a source to label those products for which it requests special handling by the IPv6 router. The flow is uniquely identified by the combination of a source address and a non-zero flow label.
- *Payload length* -- The length of payload including header.
- *Next header* -- Identifies the type of header immediately following the IPv6 header.
- *Hop limit* -- It is decremented by one by each node that forwards the packet. The packet is discarded if the Hop Limit is decremented to zero.
- *Source address* -- 128-bit address of the originator of the packet.
- *Destination address* -- 128-bit address of the intended recipient of the packet (possibly not the ultimate recipient, if a Routing header is present).

Each header type is assigned a value in IPv6. The Next Header field indicates what type of header is next in the chain by specifying its value. For example, a value of ‘50’ indicates the next header is an ESP header. The last extension header in the chain indicates that payload data is next. Additionally, IPv6 headers can be an arbitrary length and not limited to 40 bytes like IPv4. This allows options to be used for many new things which were not possible or practical in IPv4, such as authentication and security encapsulation discussed in the next section [DeH98].

2.3 Authentication Header and Encapsulating Security Payload Header

IPv6 has two significant security benefits, in the form of two extension headers. The option to use these headers is available in IPv4 for a host running the Internet Protocol Security (IPSec). However, these headers are integrated into IPv6 and are expected will gain wider acceptance and usage as IPv6 becomes a world-wide standard.

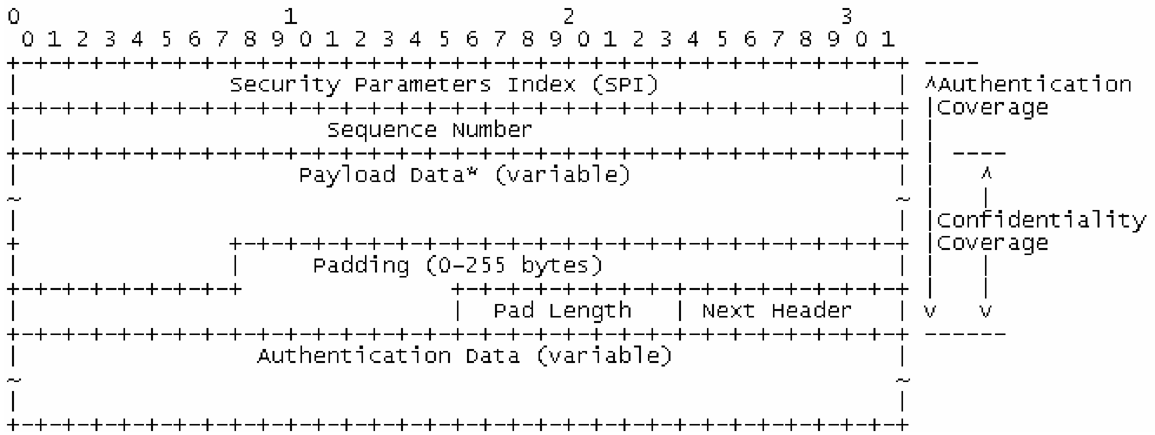
The first benefit is the Authentication Header (AH), which provides data authentication and integrity. Any data which has been tampered with en route, or generated by a spoofing source can be detected. However the data might be read by an unauthorized party [KeA98a]. This isn't a threat to an IDS, which is not concerned with authenticity of data, but rather its content.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Payload Len  |          RESERVED             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Security Parameters Index (SPI)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number Field                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                Authentication Data (variable)                 |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.3: Authentication Header (AH)

In these two headers shown in Figures 2.3 and 2.4, the *security parameter index* (SPI) field enumerates a Security Association (SA) between sender and receiver. The SA itself includes information indicating how the sender and receiver will encrypt and/or authenticate their conversations. SAs are discussed in more detail in the next section [KeA98a, KeA98b].

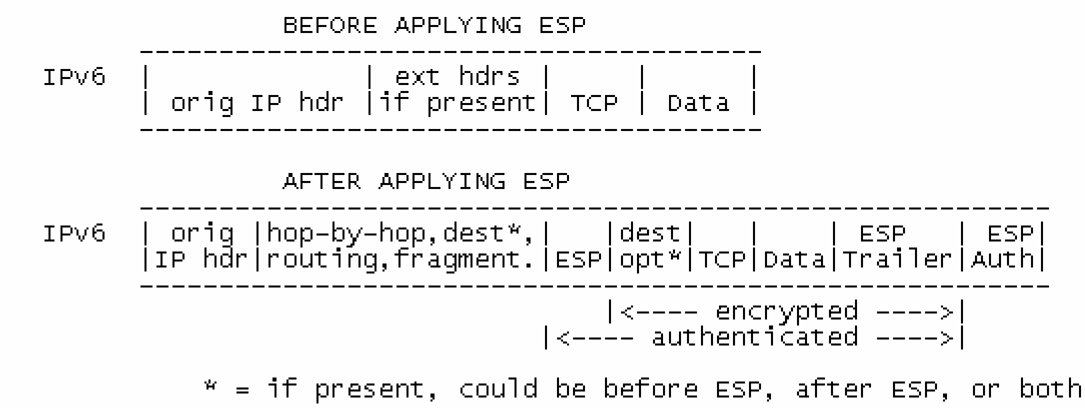


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ----
|               Security Parameters Index (SPI)                 | ^Authentication
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Coverage
|                      Sequence Number                          | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | ----
|                    Payload Data* (variable)                   | |   ^
~                                                               ~ |   |
|                                                               | |Confidentiality
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Coverage
|               |     Padding (0-255 bytes)                     | |   |
+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |   |
|                               |  Pad Length   | Next Header   | v   v
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ------
|                 Authentication Data (variable)                |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.4: Encapsulating Security Payload (ESP) Header

The *authentication data* field varies in length, based upon the authentication algorithm, and holds the authentication value computed by the sender. When a destination receives an authenticated packet, it computes an authentication value in the same way as the source, and then compares the two. If the two values match, then the packet is considered authentic.

The ESP header has a few of the same fields as the AH header with a couple of additions. The *payload* field typically contains the encrypted payload data, but can also include information required for decryption. For example, encryption algorithms such as Data Encryption Standard (DES) and Advanced Encryption Standard (AES) require an initialization vector (IV) equal in size to the block size, which is placed in the payload field prior to the encrypted data. The *next header* field indicates the type of payload which is encrypted (i.e., 6 for TCP data, or some other number if there are destination option headers included). The ESP header must be the header last in the chain as shown in Figure 2.5 as it encrypts all data following it [KeA98b]. Finally, the padding field size is variable in and used to pad the payload to a multiple of the encryption algorithm block size.



```
                BEFORE APPLYING ESP
          ----------------------------------------
    IPv6  |             | ext hdrs |      |      |
          | orig IP hdr |if present| TCP  | Data |
          ----------------------------------------

                AFTER APPLYING ESP
          ---------------------------------------------------------
    IPv6  | orig |hop-by-hop,dest*,|   |dest|   |    | ESP   | ESP|
          |IP hdr|routing,fragment.|ESP|opt*|TCP|Data|Trailer|Auth|
          ---------------------------------------------------------
                                        |<---- encrypted ---->|
                                    |<---- authenticated ---->|

              * = if present, could be before ESP, after ESP, or both
```

Figure 2.5: ESP Header Chain

AH and ESP can be used in two modes, Transport Mode and Tunnel Mode which are illustrated in Figures 2.5 and 2.6 respectively. Transport mode is generally considered a host-to-host mode. The authentication and encryption security is applied to data leaving one host, and is not removed until it reaches its destination. In this case, ultimate source and destination addresses are not protected, nor is the SPI or any header information—only payload data is protected. Tunnel mode, on the other hand, creates a

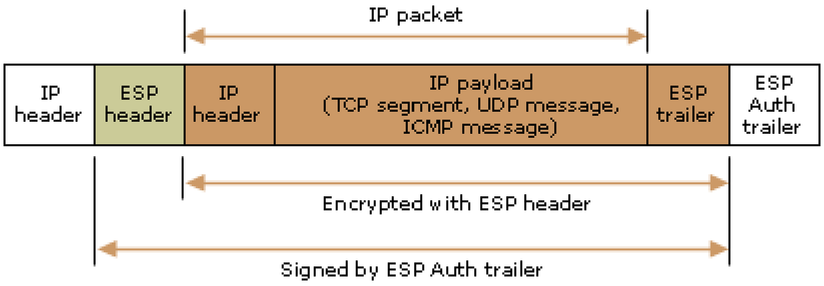

IP packet
IP header
ESP header
IP header
IP payload (TCP segment, UDP message, ICMP message)
ESP trailer
ESP Auth trailer
Encrypted with ESP header
Signed by ESP Auth trailer

Figure 2.6: ESP Packet in Tunnel Mode

secure channel between two gateways or between a host and a gateway. In tunnel mode, the inner headers (any data from the local side of the gateway) are also encrypted. If the host is tunneling, this protection is essentially useless because the outer address is the same as the inner address. This type of communication protects data from host to host, although not along the entire path—only on the length of the tunnel [KeA98a, KeA98b, KeA98c, Hui96].

2.4 Security Association Databases and Security Protocol Databases

Security Associations are the basis of IPSec functionality. Simply stated, an IPSec SA is an agreement between two nodes on a network which allows them to communicate securely. It is identified uniquely by a triple of an SPI, the destination address, and the protocol used (AH or ESP). An IPSec SA is one-way, so two-way traffic requires two IPSec SAs [KeA98c]. An SA includes an encryption/decryption algorithm, a cryptographic key, and other details such as the lifetime of the key. Keys can be updated manually by system administrators, or automatically via Internet Key Exchange routines. SAs can be set up manually, or created via the Internet Security Association and Key Management Protocol (ISAKMP) [KeA98c].

When an encrypted packet is received by a network node, it must examine the clear text SPI field, destination address, and protocol field, and correlate it with an SA from the source. If a valid SA exists, the host applies whatever rules the SA dictates to decrypt and/or authenticate the data. If an SA does not exist, the data is either discarded, or in some implementations, an SA is generated. All SAs are housed in a Security

Association Database (SAD) which resides on the endpoints of the secure channel [KeA98c].

Another important piece of the protocol is the Security Protocol Database (SPD), which correlates SPIs to SAs by specifying “what services are to be offered to IP datagrams and in what fashion.” [KeA98c]. The SPD is consulted for all incoming or outgoing traffic, and takes one of three actions: apply IPSec protection to the data, allow the data to bypass IPSec protection, or discard the packet. In the case of outgoing traffic, if the SPD determines that IPSec protection should be used, the SPD entry will contain an SA or set of SAs which should be applied. For example, a policy listing could call for “all matching traffic to be protected by ESP in transport mode using 3DES-CBC…, nested inside of AH in tunnel mode” [KeA98c]. The SPD entry is linked to a particular SA or set of SAs in the SAD, which are applied in a specific order. Similarly, with incoming traffic, the SPD is consulted, and if required, SAs are pulled from the SAD for use.

2.5 Limitations Posed Upon Network-based Intrusion Detection Systems

Network Intrusion Detection Systems (NIDS) are a popular means of protection against computer network attacks. They are placed in a subnet where they monitor network traffic. Primarily passive, they add little if any network overhead, and are virtually undetectable. Most IDSs operate similar to anti-virus programs, by searching bit strings for patterns which indicate malicious activity or network misuse. While an antivirus program examines files, a NIDS examines IP packets [Ran02].

Snort is an open-source NIDS representative of the genre. Snort is a very robust IDS which can be used on Linux, BSD or Windows platforms among others. It provides several levels of operation, from simply sniffing TCP/IP packets, to logging them, to providing full IDS service [Roe04]. Snort allows users to configure rules on which portions of traffic to examine and also provides real-time alerts to possible attacks. Though Snort is developing IPv6 support, it is not currently available in the public release. More than one effort is in place to develop IPv6 support for Snort, but it is apparently a low priority due to the rarity of IPv6 implementations. Internet Security Systems (ISS) has had a, IPv6 NIDS appliance available since 2003, but not many competitors have followed suit [Iss04].

Despite the lack of current IPv6 IDS options, future prospects are promising. There are already IPv6 packet sniffers, one such being Ethereal. Ethereal is an open source sniffer which can decode 500+ protocols, IPv6 included [Eth04]. Once the capability of sniffing and dumping IPv6 packets is in place, it is only a matter of analyzing that data and looking for signs of intrusions. *Dexter* is a proof-of-concept IPv6 IDS [HaM03] which shows that IPv6 traffic can effectively be captured and decoded for use with an IDS. Once these packets were decoded, *Dexter* classified them into different services originating the packets, and did no actual IDS analysis.

What really poses a threat to NIDS systems, and is exacerbated by IPv6, is the prospect of widespread use of the ESP capability. Typically an IP stack is transmitted in the clear, allowing a NIDS (or any intermediate source) to examine the data and do pattern matching. Encrypted packets, on the other hand, cannot be examined unless the computer capturing the packets has the decryption key and knows the decryption algorithm. This data, by design, is only available to the endpoints of the communication channel.

A solution to the problem of examining encrypted traffic with a NIDS is called a Network Node IDS (NNIDS). In a NNIDS implementation, NIDS functions are delegated to individual hosts on the network. Unlike a host-based IDS (HIDS) which examines log files and activities on a local machine, the NNIDS examines the TCP/IP traffic just as a traditional NIDS would. This arrangement has some significant advantages, such as the ability to handle encrypted traffic, and distributing the load of intrusion detection across many computers, rather than doing all packet examination with a single dedicated NIDS. The drawback, though, is each computer on the network has to maintain an up-to-date attack signature database [Nss04].

2.6 Problems to be Addressed

For a NIDS to effectively protect a network whose hosts are running IPSec, it must have enough information to decrypt and scan network traffic. In a network employing IPSec with static secret-keys, it is conceivable to load the keys onto the IDS beforehand, affording it the ability to decrypt data using those keys. IPSec, however, was intended to be more flexible than having static keys assigned to every host. Security associations can be dynamic, with keys refreshed as often as necessary for the desired level of security. To develop such a system, other support services are needed, including the Internet Key Exchange protocol and details of the algorithms used to encrypt the data.

2.6.1 Internet Key Exchange (IKE) The IKE is a framework defined in Request for Comments (RFC) 2409 for the establishment of security associations, over which IPSec transactions can take place. It is based on the Internet Security Association Key Management Protocol (RFC 2408), the Oakley Key Exchange (RFC 2412) and the Security Key Exchange Mechanism (SKEME) described below [HaC98].

> ISAKMP ([MSST98]) provides a framework for authentication and key exchange but does not define them. ISAKMP is designed to be key exchange independent; that is, it is designed to support many different key exchanges.
>
> Oakley ([Orm96]) describes a series of key exchanges—called "modes"-- and details the services provided by each (e.g. perfect forward secrecy for keys, identity protection, and authentication).
>
> SKEME ([SKEME]) describes a versatile key exchange technique which provides anonymity, repudiability, and quick key refreshment.

A “security association” of the ISAKMP is different from that in IPSec. In ISAKMP, the security association is a two-way communication path between two hosts, whereas IPSec SAs are simplex, and are negotiated using the security associations created by ISAKMP. Though both security associations ultimately accomplish the same end result (secure communications between hosts), the goal of the IKE security association is to give IPSec clients a method of creating their own SAs. Therefore, in instances of dynamic IPSec SA creation, it is likely an IKE security association precedes any IPSec SA negotiation [HaC98].

An IKE association has four modes of operation, Main, Aggressive, Quick, and Group. The association itself is handled in two phases. In Phase 1, two ISAKMP hosts establish a secure and authentic communication channel, the ISAKMP security association. Phase 1 is accomplished for both Main and Aggressive mode. Phase 2 of the ISAKMP security association is basically everything after Phase 1—it's the time when SAs are negotiated on behalf of IPSec or other services. Quick mode is used in Phase 2. In this manner, many IPSec SAs can be created with one ISAKMP security association. The remaining details about IKE operation are not relevant, because this research focuses on IPSec (Phase 2) SAs.

IPv6 and IPSec are currently available for most popular operating systems. IPSec and IPv6 are fully supported in Windows 2000 (SP1), Windows XP (SP1), and Windows Server 2003—but not concurrently. According to Microsoft's website, the Windows XP implementation of IPv6 supports IPSec with some key limitations:

> - The Authentication Header (AH) and Encapsulating Security Payload (ESP) are supported for both transport and tunnel modes. However, ESP for the IPv6 Protocol for Windows XP does not support data encryption.
> - IPSec in the IPv6 Protocol for Windows XP does not support the use of Internet Key Exchange (IKE) to negotiate security associations (SAs). IPSec policies, SAs, and the keys to calculate the Message Digest 5 (MD5) keyed hash for AH or ESP must be manually configured.
> - IPSec for IPv6 traffic is completely independent from IPSec for IPv4 traffic. IPv6 IPSec security policies are not managed with the Windows XP IPSec Policies snap–in. IPSec policies and SAs for the IPv6 Protocol for Windows XP are manually configured with the Ipsec6.exe command–line tool [Mic03]

Windows Server 2003 has similar limitations. Although it does support IPSec in IPv6, Microsoft does not recommend using the native implementation for "production use." Like Windows XP, it only provides static keying (not IKE exchanges) and although it supports the ESP headers, it does not support data encryption [Shi03].

The Unix/Linux/BSD operating systems seem more promising in this area. The KAME project started in April 1998 as an effort to integrate IPv6 and IPSec support into the BSD IP stack implementations. Researchers from Fujitsu, Hitachi, Internet Initiative Japan, NEC, Toshiba, and Yokogawa Electric joined forces and committed themselves to three or more full-time days per week working solely on the issue. Since its original two year charter, the project has been extended twice until this year, and has achieved impressive results. Current status includes support for IPv6 and IPSec as well as IKE. The current, experimental releases are available at www.kame.net, and stable products are integrated into all recent FreeBSD, NetBSD, OpenBSD, and BSD/OS releases [Kam04].

2.6.2 Encryption IPSec implementations are required to support DES-CBC, and 3DES-CBC support is recommended. DES-CBC is a combination block and permutation cipher which encrypts data in blocks of 64 bits. Although the key size is 64 bits, the effective strength of the key is only 56 bits because 8 bits are used for parity and do not contribute to key strength. In DES, the key is permuted, split, and rotated enough times to derive 16 separate keys. The plaintext is permuted in preparation for encryption, and the 16 keys are applied to the text in succession [Tro04].

For many years, beginning in 1976, DES-CBC was the government-endorsed encryption method, but was eventually broken as computers grew in power [Tro04]. In 1999, DES was broken in only 22 hours and 15 minutes by combining Electronic Frontier Foundation’s DES Cracker and a network of 100,000 computers on the Internet. Some have even suggested that for a couple of million dollars, a system can be built to crack DES in an hour [Eff04]. In 1999, 3DES was instituted, which is essentially DES three times, with an effective key strength of 168 bits.

The functionality of DES and 3DES is straightforward, and many implementations can be found in applications. Additionally, comprehensive cryptographic libraries are easy to obtain on the Internet for performing this encryption and decryption. Since these are private symmetric key encryption schemes, their security lies solely in the security of the key itself. Therefore, any method of sharing keys must afford them the same security that they are in place to provide.

One of the challenges of this research is finding a scheme that shares secret keys efficiently over a LAN. A LAN running IPSec/IKE clearly has a method of secure communication between the IDS and individual hosts. Considering the NIDS as just another node on the network, hosts are able to establish security associations with the NIDS and use the link to transfer keys for their other SAs. This effectively replicates the SADs of every network host on the NIDS, except unnecessary information is stripped off. For example, the NIDS need not be concerned with authentication of a packet, and only needs to know how to decrypt traffic destined for a particular host, and what key to use in its decryption. It is important for the NIDS and host to authenticate with each other, and IKE provides for this authentication.

2.7 Relevant Research

There are numerous developments taking place with IPv6 as it transitions from RFCs to actual implementations. Most popular operating systems including BSD and Windows include support for at least the basic features of IPv6. The IPSec portion of IPv6 is not as widely implemented, though it is being developed. KAME is one example of this research. According to www.kame.net, they have integrated IPv6 into BSD with

IPSec support, “good coverage of algorithms on RFC,” and an IKE implementation called *racoon* [Kam04].

Research of security concerns associated with IPv6 is also plentiful [War03, Hei04, CoM04], though not all directed at IDSs. *Dexter* is one example of such work, but there are others. Snort developers are integrating IPv6 integration into their NIDS, but a working version is not available at this writing. Current IDS systems are concerned with Secure Socket Layer (SSL) or Transport Layer Security, and some solutions have been postulated “from session key sharing by the web server allowing ‘on-the-fly’ decryption to server-side storage of keys for later off-line decryption of packets” [Tri03]. With the advent of ESP built into IPv6, the problem worsens. Encrypted traffic becomes invisible to the NIDS, and there is no way to ensure it is not malicious.

One development in the commercial sector comes from McAfee [Mca04]. The Intrushield 2600 is an IDS appliance which protects encrypted traffic for SSL. SSL is a method of data encryption for web-sessions which uses public-key encryption to establish shared session keys with a client (typically a web browser) through a handshaking process [Mca04]. By duplicating the server’s private key on the IDS appliance, the IDS can monitor the handshake and obtain the shared key. Thus possessing the shared key, the IDS can decrypt and examine future network traffic for that SSL session.

Likewise, there is an abundance of research into IDS and NIDS systems in general. Perhaps a relevant idea is the notion of the NNIDS [Nss04], mentioned earlier. NNIDS systems approach the problem of encrypted traffic on two fronts: in one they can examine encrypted traffic because the NNIDS resides on the machine where the traffic is decrypted; in the other NNIDSs increase the amount of traffic which can be effectively scanned by distributing it across many machines. The latter is the most likely hindrance to the success of a secret-key sharing framework. Sharing keys uses network bandwidth, and maintaining a key database on an already heavily tasked NIDS computer requires precious CPU cycles--both leading to an ultimate slowdown of the NIDS and reduced performance. Clearly, as processing time increases, the load a NIDS can handle decreases.

2.8 Summary

This chapter introduced IPv6 and IPSec concepts which are important to the research. Support for IPv6 is not at the same level as IPv4 support. Although this research project encompasses several different aspects of Internet security, the slow implementation of IPv6 may prove to be the biggest impediment.

3 Methodology

3.1 Introduction

This chapter presents the experimental methodology used for the research. Section 2 describes the system under test. Section 3 defines the system services, and Section 4 describes the workload presented to the system. Section 5 explains the metrics which are observed in the experiments. Sections 6 and 7 explain the factors and parameters. Section 9 describes the test bed setup for the experiments, and Sections 10 and 11 explain the evaluation technique and experimental design. Section 11 describes how the results will be used, and Section 12 is a summary of the chapter.

3.2 System Boundaries

The System Under Test (SUT) for this research consists of the secret-key sharing framework and the attached intrusion detection engine which provides the metric of primary interest. The component under test (CUT) is the secret-key sharing framework itself, which consists of secret-key sharing processes on each host, as well as a central node (co-resident with the IDS software) which collects keys in a database, and captures and decrypts packets.

The scope of the experiments is limited in several regards. To begin with, IPv6/Ethernet traffic is the only combination allowed, as the IDS interface is only written to decode such traffic.

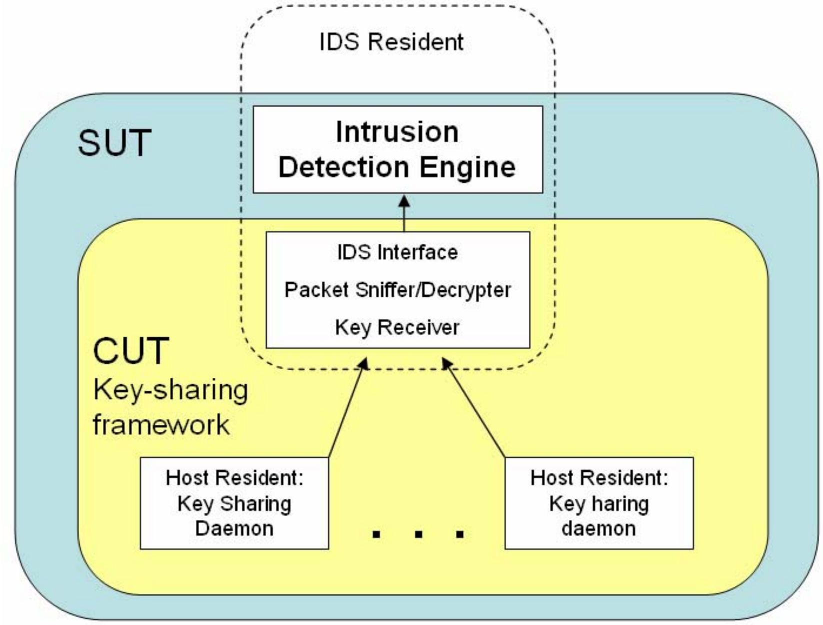


Figure 3.1: System Under Test

Another limitation is the IDS itself. The custom IDS used in this research is a basic pattern matching engine. Overall performance of a particular IDS depends on the composition of its rule set and its detection methods. The nature of this experimental research doesn’t dictate that a state-of-the-art IDS be used. The throughput of the generated workloads is not especially high. Packet interarrival times and burst distribution shape parameters are varied to ensure that IDS is capable of keeping up with the offered load. Although the numbers used may not mirror practical system operation, there is little to be gained from the experiments if the IDS cannot reach a steady state of packet processing.

Finally, the test bed is relatively small due to resource limitations with the test LAN consisting of six hosts and one IDS machine.

3.3 System Services

The system provides an intrusion detection service on the encrypted workload. The possible outcomes of this service are that genuine attack attempt was detected, or the IDS gave a false positive. A false positive is indicative of a more fundamental problem, as packet content is carefully controlled to avoid such an occurrence. In addition to generating an alert on a genuine attack or false positive, the IDS could fail to detect an attack altogether. This might be due to excessive packet drops (whether or not the IDS CPU is fully utilized), or a failure of the secret-key sharing framework to send the appropriate key to the IDS for ESP packet decryption.

Over the duration of each experiment, the number of attack detections is observed. A high percentage of detections relative to the number of attack attempts indicates that the framework is functioning properly.

3.4 Workload

The workload for the SUT is not based on an entirely realistic LAN workload. A "typical" LAN workload characterization is difficult due to the large number of possible LAN configurations. Instead, several workloads are used to provide diverse inputs to the system to discover behavioral trends. Through a series of pilot studies, the maximum throughput (or maximum offered experimental load) for each workload is determined by noting when the NIDS starts missing a significant number of attacks (more than approximately 5%, which indicates a failure of the secret-key sharing framework). All traffic is generated at the application layer, and each TCP payload contains an attack signature.

The offered workloads vary in three regards. First, the percentage of network traffic which is encrypted varies from 0-100%. Although it is unlikely for a network to see 100% encrypted traffic, that level is used to stress the framework. The second variation is the payload size. Payload size distribution has a significant effect on network performance in terms of "good" data throughput. The workloads for these experiments vary from a small payload size to nearly maximum payload size, including a statistical mix. Finally three network configurations are used to alter traffic patterns. Table 3.2 in Section 3.7 contains a tabular listing of workload characteristics and the associated levels.

These workloads are justified by the goals of the research. The goal is *not* to determine how a particular IDS performs in a particular environment, but rather to determine if the secret-key sharing framework adds significant overhead to the network and IDS CPU under various conditions.

3.5 Performance Metrics

The SUT provides several metrics which are of interest. The first is the number of attack signatures detected over a period of time, or the derived metric of attack detection rate (number of attacks detected/number of attacks attempted). This is clearly important as it is the primary indication the system is functioning. Beyond its use to derive useful offered load data rates, the metric is not analyzed. Dropped packets are also measured, although this metric is primarily used to determine appropriate workload characteristics.

In addition to attack detection rate, a record is maintained of how many packets could not be decrypted. This metric, in conjunction with the number of dropped packets, is valuable in determining the cause of variation in the attack detection rate.

The next two metrics are network throughput and network goodput. In this case, “goodput” refers to the amount of “good” or “user” data which is being sourced to the SUT. Although throughput is heavily dictated by the offered load, an accurate measurement of data presented to the SUT is beneficial for drawing conclusions about dropped packets and overloading the system. Additionally, the level of network overhead created by the secret-key sharing framework is determined.

The last metric is the CPU utilization of the computer hosting the IDS and key-receiving suite.

3.6 Parameters

3.6.1 System Parameters All hosts are running the FreeBSD 4.10 operating system as a VMWare guest on a Windows 2000 host machine. FreeBSD was chosen due to its unique support for IPv6, IPSec, and Encapsulating Security Payload headers. No other available operating system supports this combination of requirements (with the exception of other *BSD releases). The decision to operate as a VMWare guest was driven by the convenience of the setup, and ability to easily make system changes and revert to previous configurations in case the installation was broken or corrupted during the research effort. Though this causes some performance penalty, it will be a penalty across the board on all configurations and therefore is not significant.

Parameters specific to the secret-key sharing framework are defined as follows. Though the Phase 2 (IPSec) key lifetime is varied, the Phase 1 (IKE) key lifetime is set to 1 minute. This generates a good amount of activity on the framework with Phase 1 key exchanges, and prevents a false indication of steady state due to keys being static for a long period of time.

Encryption type is a parameter of both the system and the workload. The encryption protecting the secret-key sharing data is the same type and strength used to encrypt the workload. In these experiments, encryption type is set to triple DES (3DES). DES is required by the Internet Key Exchange (IKE) RFC, 2409 and 3DES is recommended. This and its status as one of four FIPS approved encryption algorithms makes 3DES a well-founded selection [Nis04].

Finally, subnet size is a parameter which remains constant for the experiment. It is set to six hosts, plus the NIDS. Although this is small for a realistic model, it is sufficient to stress test the secret-key sharing framework in the desired way. Subnet size certainly affects performance, but it is not examined in this research.

<u>3.6.2 Workload Parameters</u> There are a number of workload parameters which affect the performance of the secret-key sharing framework. Attacks are generated in a bursty fashion, according to a Pareto distribution with shape parameter of 1.5. A shape parameter between 1.5 and 2.0 is appropriate for self-similar network traffic, and converges within a finite number of samples [Bal99, Kra04]. All three attack payload

sizes mandate slight adjustment of the workload parameters so that the highest possible throughput is used regardless of this factor. For all scenarios, time between bursts are exponentially distributed, and the minimum inter-payload wait times are constant, with values listed in Table 3.1. The inter-burst wait time and inter-payload wait times were determined through pilot studies to stress the IDS CPU without overloading the system to the point of dropping a high percentage of packets (greater than 5%)

Table 3.1: Inter-burst and Inter-Packet Wait Times

Payload Size	Inter-burst Time (ms)	Inter-payload Time (ms)
Small	500	10
Mix	500	10
Large	500	100

3.7 Factors

The five factors varied include key method, Phase 2 key expiration time, network configuration, payload size, and percentage of encrypted load. Table 3.2 contains the levels for each factor, and the sections following the table describe the levels in detail.

3.7.1 Percentage Encrypted Traffic Percentage encryption is varied 0%, 33%, 66%, and 100%. These levels provide some granularity which can be analyzed, rather than just 0 and 100% encrypted traffic. Varying the percentage encryption causes more or less activity on the IDS-resident SA database.

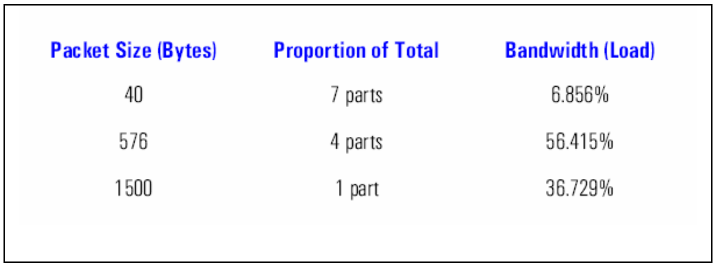
Table 3.2: Factors and Levels

	Factor	Levels
Workload Characteristics	% Encrypted Traffic	0%, 33%, 66%, 100%
	Payload Size	small, large, statistical mix
	Configuration	Star (6-to-6) Single (3-to-3) Server (5-to-1)
Other Factors	Key Method	static, dynamic
	Phase 2 Key Expiration Time	15 seconds, 30 seconds

3.7.2 Payload Size Packet size is not directly controlled. Attack payloads are generated at the application level, leaving the TCP/IPv6 stack to encrypt, encapsulate, and transmit them. This is needed to ensure the framework functions with the IPSec enabled FreeBSD stack. As a result, packets cannot be directly crafted. Thus, true packet size is a factor of the TCP/IPv6 stack as well as encryption constraints, and payload size is varied to generate packet near to the desired size. Payload size, then, takes on one of three values: small, large, and statistical mix.

The small payload size is a very small payload containing the attack signature padded to 40 bytes. Maximum sized payloads contain an attack signature and are padded to about 1350 bytes which allows 40 bytes of header IPv6 header data, 20 bytes of TCP data, and room for the ESP header/padding without exceeding the maximum transmission unit of 1500 bytes for an Ethernet packet. The statistical payload mix is based on the assortment shown in Table 3.3. The mix is derived from Internet data collected in February 2001 by the Measurement & Operations Analysis Team from the NLANR (National Library for Applied Network Research) project, as analyzed in [Agi04]. In this set of experiments, the statistical mix is interpreted as the payload size, rather than the packet size. Thus, the statistical mix workloads contain 6 parts 40 byte attacks, 4 parts 576 byte attacks, and 1 part 1350 byte attack.

Table 3.3: Packet Size Statistical Mix



Packet Size (Bytes)	Proportion of Total	Bandwidth (Load)
40	7 parts	6.856%
576	4 parts	56.415%
1500	1 part	36.729%

3.7.3 Network Configuration In lieu of varying the subnet size, three network configurations are examined with fewer to more required SAs. In the Single configuration, three unique peer-to-peer connections are used, requiring six SAs. In the Server configuration, one host acts as a server, or central node, to which the other five hosts send attacks requiring 10 SAs. Finally, the Star configuration generates traffic from each host to a random selection of the 5 other hosts, which requires up to 30 SAs. Figure 3.2 illustrates the different configurations.

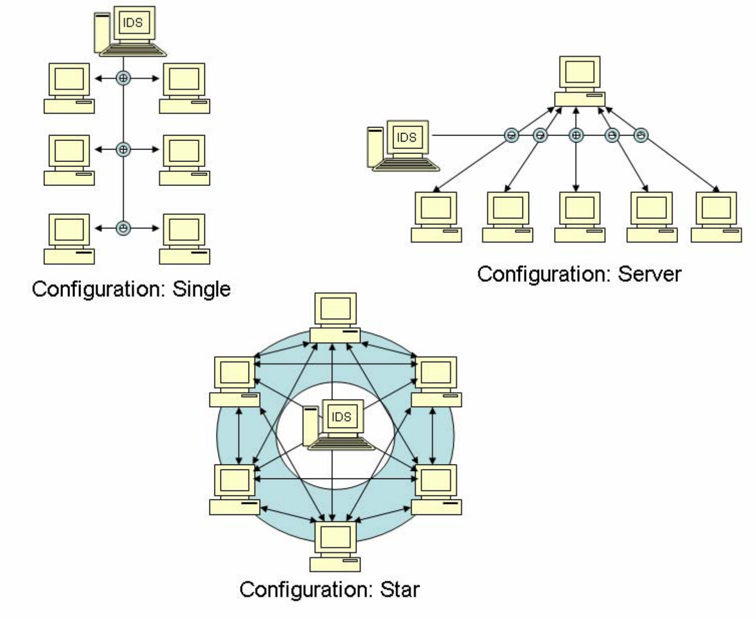


Figure 3.2: Network Configurations

3.7.4 Key Method The keying method is either static or dynamic. Static keying creates the baseline results which are subsequently compared to dynamic keyed results. With static keying, all keys are provided beforehand to each host, and a list of those keys is also provided to the decryption processor. In dynamic mode, network hosts generate IPSec keys dynamically through IKE (using a modified version of *racoon*), then update those keys and send them to the decryption processor, adding both network and CPU overhead to the hosts and IDS machine.

3.7.5 Phase 2 Key Expiration Time Also driven by the LAN size limitation, Phase 2 key expiration time is set to two fairly short values, 15 and 30 seconds, generating high levels of key-exchange traffic.

3.8 Test Bed Setup

This section describes all vital information needed to recreate the experiment. Sections 3.8.1 and 3.8.2 describe the physical setup and IPv6 / IPSec configuration, respectively, as well as the software present on each network host. Sections 3.8.3 through 3.8.7 provide more detail on the software packages and their setup.

3.8.1 Physical Setup The entire test LAN consists of 8 separate physical computers. Each computer is running Windows 2000 with VMWare 4.01 build 5289 executing a FreeBSD 4.10 guest OS. All computers are Dell Poweredge 1650's, with the exception of the IDS, which is a Dell Poweredge 1750. Specifications and software packages are provided in Table 3.4. Computers are connected through an 8 port hub and limited (via VMWare) to 10Mbps.

Table 3.4: Computer Configuration

Machine	**IPv6 Address**	**CPU**	**Installed Memory (allocated to VMWare)**	**Research Software Installed**
IDS	2004::1	Dual Intel P4 2.4 GHz*	1024 (768)	server cpumon racoon
Generic Host	2004::2 2004::3 2004::4 2004::5 2005::7 2004::8	Intel PIII 1.4GHz	512 (384)	attackclient v6listen cpumon racoon2
Control Node	2004::6	Intel PIII 1.4GHz	512 (384)	attacknet

* The VMWare guest OS only uses one processor

3.8.2 IPv6 / IPSec Setup Enabling IPv6 in FreeBSD 4.10 is a simple matter of adding a few lines to the *rc.conf* file, since the kernel enables IPv6 support by default. In these experiments, no host was acting as an IPv6 router, all addresses were set manually.

```
# file: rc.confl
ipv6_enable=”YES”                #enable IPv6
ipv6_ifconfig_lnc0=”2004::1”     #configure network device lnc0 with IPv6 address 2004::1
```

Configuring IPSec begins with a few non-default kernel options. The IPSec-enabled kernel must include the following lines in the configuration file:

```
# file: mykernel
options        IPSEC              #IP security
options        IPSEC_ESP          #IP security (crypto; define w/ IPSEC)
options        IPSEC_DEBUG        #debug for IP security
```

After building the kernel with this configuration, IPSec has to be implemented via SPD entries, as well as SAD entries in the case of manual (static) keying. Both are accomplished with the *setkey* utility. For either static or dynamic keying, SPD entries have to be generated. This is a matter of defining a set of host-host policies along with associated processing conditions. First, all ICMPv6 traffic is explicitly excluded from IPSec processing. Unfortunately, IPv6’s neighbor solicitation/discovery takes place via ICMPv6 and IPSec protection of such traffic leads to hosts “losing” each other on the network.

```
#file: manual_1 or racoon_1.15 or racoon_1.30
#! /bin/sh
setkey -c <<EOF
spdadd 2004::2 2004::1 ipv6-icmp -P out none;   #exclude incoming icmpv6 traffic
spdadd 2004::1 2004::2 ipv6-icmp -P in none;    #exclude outgoing icmpv6 traffic
… (repeat for each host)
EOF
```

To distinguish between encrypted and clear traffic using IPSec security associations, two ports are used—port 326 for encrypted traffic, and port 325 for clear traffic. Security Policies are set up to encrypt/decrypt any outgoing/incoming traffic according to IPSec-ESP rules. Note port 525 is used for secret-key sharing with the IDS interface, and is therefore protected with IPSec as well.

```
#file: manual_2 or racoon_2.15 or racoon_2.30
#! /bin/sh
setkey -c <<EOF
spdadd 2004::2 2004::1[525] tcp -P out ipsec esp/transport//require ;   # process outgoing
spdadd 2004::1 2004::2[525] tcp -P in ipsec esp/transport//require ;    # and incoming
spdadd 2004::2[525] 2004::1 tcp -P out ipsec esp/transport//require ;   # data on port 525
spdadd 2004::1[525] 2004::2 tcp -P in ipsec esp/transport//require ;    # to/from host
                                                                        # 2004::1
spdadd 2004::2 2004::3[326] tcp -P out ipsec esp/transport//require ;   # process outgoing
spdadd 2004::3 2004::2[326] tcp -P in ipsec esp/transport//require ;    # and incoming
spdadd 2004::2[326] 2004::3 tcp -P out ipsec esp/transport//require ;   # data on port 326
spdadd 2004::3[326] 2004::2 tcp -P in ipsec esp/transport//require ;    # to/from host
                                                                        # 2004::3
… (repeat for each host)
EOF
```

At this point, the configuration files diverge for static and dynamic keys. When using static keys each SA is manually entered, including encryption key. This is accomplished through the following statements. Although symmetric keys are used, each two-way association requires two SAs and therefore two keys.

```
#file: manual_2
#! /bin/sh
setkey -c <<EOF
#add spi 2121 to the SAD:
#traffic from 2004::2 to 2004::1 should be encrypted with 3des-cbc, using the key provided
add 2004::2 2004::1 esp 2121 -E 3des-cbc
0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa;
#add spi 1212 to the SAD:
#traffic from 2004::1 to 2004::2 should be encrypted with 3des-cbc, using the key provided
add 2004::1 2004::2 esp 1212 -E 3des-cbc
0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef;
… (repeat for each host)
EOF
```

When using *racoon* or *racoon2* IKE daemons, the daemons simply need to be executed with a provided configuration file.

```
#file: racoon_2.15
#! /bin/sh
/usr/local/sbin/racoon2 -f /usr/v6/racoon/racoon15.conf    #execute racoon2 using racoon15.conf
```

The *racoon.conf* file outlines many configuration options for the racoon or racoon2 daemons. Most defaults were not changed from the example *racoon.conf* included with the *racoon* packages from KAME. For these experiments, the IKE authentication method used was “pre-shared key,” and keys were stored in *psk.txt* (set to mode 600). The important non-default configuration lines for *racoon.conf* are shown below. All settings were identical for each host, guaranteeing the ability to communicate.

```
#file: racoon15.conf
path pre_shared_key "/usr/v6/racoon/psk.txt" ;
#IKE configuration
remote anonymous
{
        exchange_mode aggressive,main;
        my_identifier user_fqdn "sakane@kame.net";
        nonce_size 16;
        lifetime time 1 min;                                # sec,min,hour
        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method pre_shared_key ;          #authentication method
                dh_group 2 ;                                #use diffie-hellman group 2 for key
generation
        }
}
#IPSec configuration
sainfo anonymous
{
        pfs_group 1;                                        #perfect forward secrecy group 1
        lifetime time 15 sec;                               #key lifetime = 15 seconds
        encryption_algorithm 3des ;                         #encrypt with 3des-cbc
        authentication_algorithm hmac_sha1;                 #use hmac_sha1 for authentication
        compression_algorithm deflate ;
}
```

3.8.3 *racoon* and *racoon2* *Racoon* is an IKE daemon developed by the KAME project [Kam04]. The version used here is racoon-20040408a. *Racoon2* is a slightly modified version of *racoon*, set up to send any freshly generated keys, as well as key expiration messages, to the IDS interface. Modified/new files for this research include:

Filename	(M)odified or (N)ew	Functionality Affected/Added
pfkey.c	M	isakmp_ph2delete(iph2): now invokes SAexp
isakmp.c	M	pk_sendupdate(iph2): now invokes SAadd pk_sendadd(iph2): now invokes SAadd pk_recvexpire(mhp): now invokes SAdel
SAadddel.c	N	SAadd: sends SA information to IDS interface SAdel: informs IDS interface that an SA has been deleted
SAexp.c	N	SAexp: informs IDS interface when an SA has expired

When a message needs to be sent to the IDS, a thread is started and detached, limiting the overhead on the host machines. As such, if the message send fails there is no error checking to that effect. Messages take the formats:

Message Type	Format
Add a Key	ADD^<source>^<destination>^<spi>^<encryption type>^ <encryption key length>^<authentication type>^<authentication field length>^ <encryption key>^<lifetime>^<life bytes>^
Delete a Key	DEL^<source>^<destination>^<spi>^
Expire an SA	EXP^<source>^<destination>^

These update messages are sent over port 525 and are encrypted using IPSec, guaranteeing their confidentiality to the level of the keys they are transmitting.

3.8.4 *v6listen* A multi-threaded listener for TCP/IPv6 which listens to ports 325 and 326 on each host computer.

3.8.5 *server* This is the primary application under test. *Server* has many threads of operation with the functions of accepting key updates from hosts, capturing and decrypting network traffic, and performing intrusion detection on the sniffed packets. In general terms, *server* maintains an IDS-resident copy of all SADs on the network. It then uses the data to decrypt ESP packets on the network and perform intrusion detection. Figure 3.3 shows a detailed flow diagram of *server* functionality.

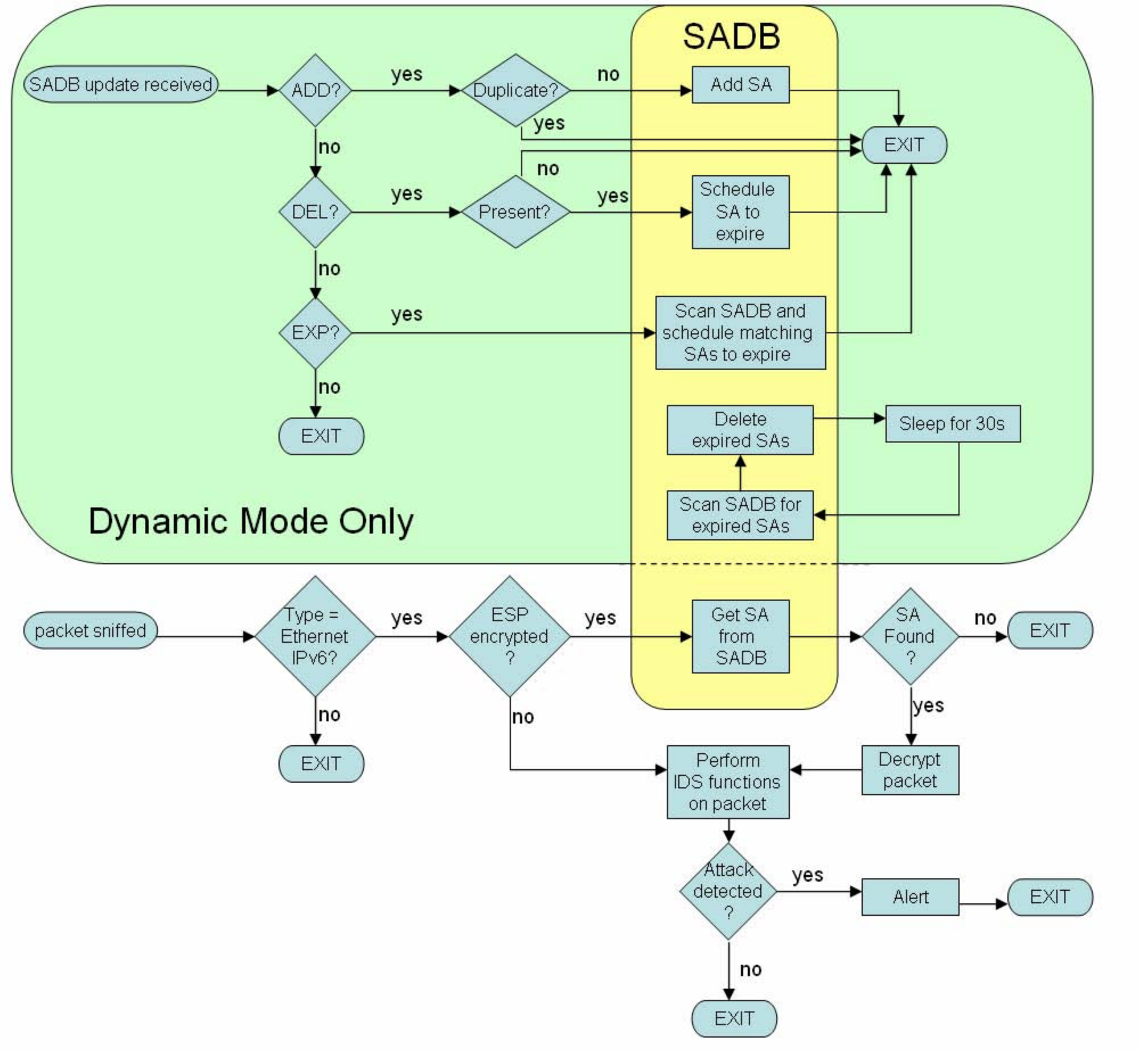


Figure 3.3: *server* Functionality

Server operation is dependent on several publicly available libraries:

- *libpcap 0.8.1*: used by many popular applications such as Snort and tcpdump, *pcap* is an integral part to any packet-sniffing application [Pca04].
- *libpthread*: posix thread system, used for implementing *server*'s threads of execution.
- *libbotan v1.2.8*: *botan* is a cryptographic library selected for its ease of use and robust support for many algorithms. Though *server* natively only supports DES, Triple DES, and AES, expanding support to other IPSec algorithms such as cast128 and blowfish is trivial, making *botan* a good selection.

The three primary threads of *server* are shown in Figure 3.3 and include the packet-capturing thread, the SAD maintenance thread, and the SAD update thread, which is actually a pool of 32 threads receiving updates. When running in static keying mode, the SAD update threads and SAD maintenance threads are not started, leaving only the packet sniffing thread running.

3.8.6 *attacknet* and *attackclient* To test the secret-key sharing framework, an network of attacking computers was developed consisting of an instance of *attackclient* running on each host, and one master node running *attacknet* which controls the clients. Each client is initialized with the IDS rules file, and uses those rules to create and send attack payloads to another listening host. All attacks begin at the application layer, by writing attack payloads to an open socket. These clients maintain counts of the number of attacks sent, as well as the number of bytes sent during a particular run of an experiment.

Attacknet is the master control for the experiments. It maintains a connection to each host, as well as to the IDS. Prior to an experimental run, *attacknet* polls and logs starting status of the clients, resets their counters, and sets the traffic generation parameters including random seed. It then tells the IDS to begin logging packet data, and begins attacks. Upon completion, *attacknet* halts the attacks, logs end status of the clients as well as attacks detected by the IDS. Although different modes can be set while *attacknet* is running, it must be provided a configuration file for startup which tells *attacknet* the address of the IDS and the clients on the network, as well as provides the option to load up to 64 quick-command buffers. If complex commands are run often, they can be loaded in the configuration file and then quickly executed by typing "quick <number>." An example of a configuration file for running in star mode is shown in the following box.

```
#file: attacknet-star.cfg
#first line is the address of the IDS
2004::1
#these are the clients
2004::2 2004::3 2004::4 2004::5 2004::7 2004::8
#following are quick cfg lines:
timesync
connect 600
status
2004::2 -cfg star 2004::3 2004::4 2004::5 2004::7 2004::8
2004::3 -cfg star 2004::2 2004::4 2004::5 2004::7 2004::8
2004::4 -cfg star 2004::3 2004::2 2004::5 2004::7 2004::8
2004::5 -cfg star 2004::3 2004::4 2004::2 2004::7 2004::8
2004::7 -cfg star 2004::3 2004::4 2004::5 2004::2 2004::8
2004::8 –cfg star 2004::3 2004::4 2004::5 2004::7 2004::2
```

3.8.7 *cpumon*: This is a simple utility which reads the FreeBSD KVM to determine how the CPU is being utilized. The output is directed to a file which is then parsed to determine CPU utilization over time. A sample of the output is:

```
#file: xx-xx-icpu
Time        User        Wait        Kernel      Int         Idle
1102109297  0.000000    0.000000    0.031250    0.000000    0.968750
1102109298  0.007752    0.000000    0.023256    0.000000    0.968992
1102109299  0.010336    0.000000    0.023256    0.000000    0.966408
1102109300  0.009690    0.000000    0.023256    0.000000    0.967054
1102109301  0.009288    0.000000    0.023220    0.000000    0.967492
```

3.9 *Evaluation Technique*

These experiments are conducted through empirical study. The availability of the FreeBSD operating system with nearly all desired functionality makes the choice fairly simple. The combination of *cpumon* and *server* provide all of the necessary metrics to evaluate the system. Additionally, all hardware for empirical study is readily available. Evaluation with analysis or simulation would likely require as much or more effort.

Validation of the test bed is conducted in several steps. As the framework is being built essentially from scratch, testing is done throughout development to ensure correct operation of different components. When all sub-parts are integrated into the finished product, pilot studies and single attacks are executed to verify overall functionality. The steps taken to validate the test bed are listed in Table 3.5.

Generic validation criteria is that single network attacks are decrypted and decoded correctly by IDS interface and the appropriate alert is generated by the IDS. Using two clients, one to send and one to receive, a single attack is generated with the framework in dynamic mode, and then in static mode. If the IDS flags an alert, the

Table 3.5: Validation Tests

Validation Goal	Validation Test	Validation Criteria
Detect single encrypted attacks	1 attack sent from client to client	Attack is decrypted by the IDS interface, and the IDS alerts the attack
Detect multiple encrypted attacks	100 attacks sent from each host in star configuration at a low rate in both static and dynamic modes	All attacks detected
server sustains proper operation	300 second runs in star configuration with low attack rate in dynamic mode	*server* SAD size increases and decreases appropriately when scenario is over
Framework generates appropriate traffic on network	300 second runs in star configuration are examined in Ethereal	TCP conversations between the IDS and clients only occur at specified key expiration intervals

framework is functioning correctly. Then, a test of multiple attacks is conducted. A set number of attacks are sent at a slow rate to ensure all attacks are alerted. Finally, attacks are sent for 300 seconds to ensure that *server* is functioning correctly for multiple key exchanges, and maintaining the SAD appropriately while not letting it grow out of control before deleting expired keys. Finally, traffic generated by the framework is examined with Ethereal to ensure it is not generating more traffic overhead than it should be. Once the test bed is validated, empirical data from the experiments can be considered valid as well. Pilot studies are conducted, as mentioned in Section 3.6.2, to determine traffic parameters for which there is a low to zero level of packets dropped by *server* for each network configuration and packet size.

3.10 Experimental Design

A full factorial experiment is conducted. For each network configuration and packet size, the baseline is the IDS CPU utilization with *server* running in static key mode with 0% encryption. These and all other experiments are shown in Table 3.6.

Table 3.6: Experiment List

Configuration	Key Method	Key Expiration	% Encrypted	Payload Size
Star	Static	n/a	0,33,66,100%	small, mix, large
	Dynamic	15, 30 sec	0,33,66,100%	small, mix, large
Single	Static	n/a	0,33,66,100%	small, mix, large
	Dynamic	15, 30 sec	0,33,66,100%	small, mix, large
Server	Static	n/a	0,33,66,100%	small, mix, large
	Dynamic	15, 30 sec	0,33,66,100%	small, mix, large

Each experiment is executed until steady state is reached, i.e., when the standard deviation of the IDS CPU utilization is within about 5% of the mean. Ten replications with different random seeds are executed for each experiment. It is important to note that repeating an experiment twice with the same seed is likely to produce slightly different results. Although burst sizes and generated wait times are consistent between runs, actual wait time is difficult to control. Initially, the *usleep(long microseconds)* was used for delaying such intervals. However, due to the fact that it suspends thread execution and does not guarantee a return time, a new usleep function was created to simply occupy the processor with a busy loop until the specified number of microseconds had passed. The maximum resolution required by the experiments is around 10 milliseconds, and a comparison between the accuracy of *usleep* and *new_usleep* at 10 milliseconds (Table 3.7) shows how much more reliable *new_usleep* is. Even this does not guarantee exact results, but provides higher consistency by maintaining control of the CPU.

Table 3.7: *usleep*/*new_usleep* Comparison

Function	Requested Wait (μs)	Replications	Actual Wait (Mean)	Actual Wait (Standard Deviation)
usleep	10000	10000	19984.73	5069.38
new_usleep	10000	10000	10036.99	304.72

3.11 Experimental Design for Results Analysis and Interpretation

Upon completion of the experimental trials, the data is compared to the baseline data, and the CPU utilization examined across runs from least CPU intensive to most. It is determined whether the secret-key sharing framework imparts statistically significant overhead onto the network and/or IDS CPU. Analysis of Variance (ANOVA) is performed across the factors of key expiration time, percentage encrypted traffic, and keying mode to determine how each affects the CPU utilization and throughput. It is expected that main effects will explain the vast majority of variance, and interaction effects will be insignificant. The results of these tests are detailed in Chapter 4.

3.12 Summary

This chapter presents an experimental methodology for determining the effect of a secret-key sharing framework on intrusion detection capability for encrypted traffic and the overhead it imparts on the network and IDS CPU. Experimental results and analysis is presented in Chapter 4.

4 Analysis

4.1 Introduction

This chapter presents experimental results and analysis. Section 4.2 describes how the data was collected and compiled. Section 4.3 discusses the data rate selection, and goodput/throughput ratio. Section 4.4 explains attack detection rates and Section 4.5 looks at the network overhead directly related to the IDS. Section 4.6 examines the CPU utilization for the IDS computer. Section 4.7 discusses research limitations, and Section 4.8 is the chapter summary.

4.2 Data Collection and Analysis Methods

The experiments have ten trials per iteration, with each trial is lasting for five minutes. This allows the standard deviation of IDS CPU utilization to fall to within approximately 5% of the mean. Packet logs are collected by the IDS computer for data rate analysis. The IDS, rather than a third-party computer, is chosen to measure throughput so that decrypted goodput can also be logged. The *server* software also collects the IDS CPU utilization measurements and all statistics relating to packet collection/decryption/attack detection. Time between experiments is at least 45 seconds which allows IPSec keys to expire and require refreshing. This also gives an opportunity to observe transient behavior.

Data is initially compiled using a custom C++ program called *parsedata*. This program extracts mean values for IDS and host CPU utilization, throughput, and goodput, as well as raw values for attack detection rate, *pcap* packet drop rate, and number of undecryptable packets. The data imported into Excel where it is further consolidated and displayed in graphical form. Analysis of Variance (ANOVA) for the IDS CPU response and throughput is performed with Mathematica. Mathematica provides native support for three-way ANOVA and is discussed in Section 4.6.

4.3 Throughput and Goodput Rates

In an environment of gigabit Intrusion Detection Systems, the most obvious shortcoming of these experiments is the low data rates of the presented workloads. Figures 4.1-4.3 display the experimental mean throughput rates, grouped by attack payload size. As shown, the throughput ranges from about 0.04Mbps to 0.35Mbps. Given the maximum throughput allowed by VMWare is 10Mbps, the submitted throughput is quite low. A few factors influence the restriction. To begin with, the aggregate traffic generated by six hosts generating attacks with no delays between is about 2.5Mbps. However, the limiting factor is the *server* software, which was dropping greater than 50% of packets at that data rate, the cause of which is discussed in the next paragraph. Consequently, the attack generation rate is reduced until an acceptably low percentage of packets are dropped by the IDS computer, as discussed in Section 3.6.2.

The cause of the high packet drop rate is ultimately undetermined, but there are a few possibilities. Surprisingly, it is not due to the increased loading placed on the CPU by the decryption process. It also is not due to the un-optimized detection engine. Pilot studies which only logged packet arrivals and drops without decryption or intrusion detection functions yield similarly high packet drop rates. Additionally, since the

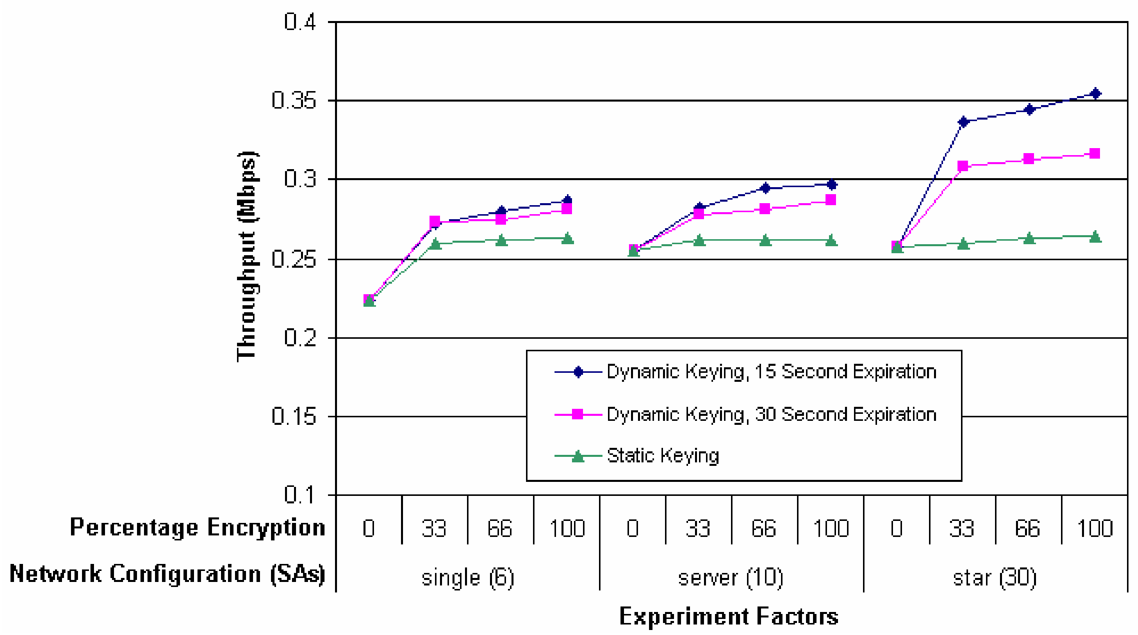


Figure 4.1: Mean Throughput for Experiments with Large Attack Payloads

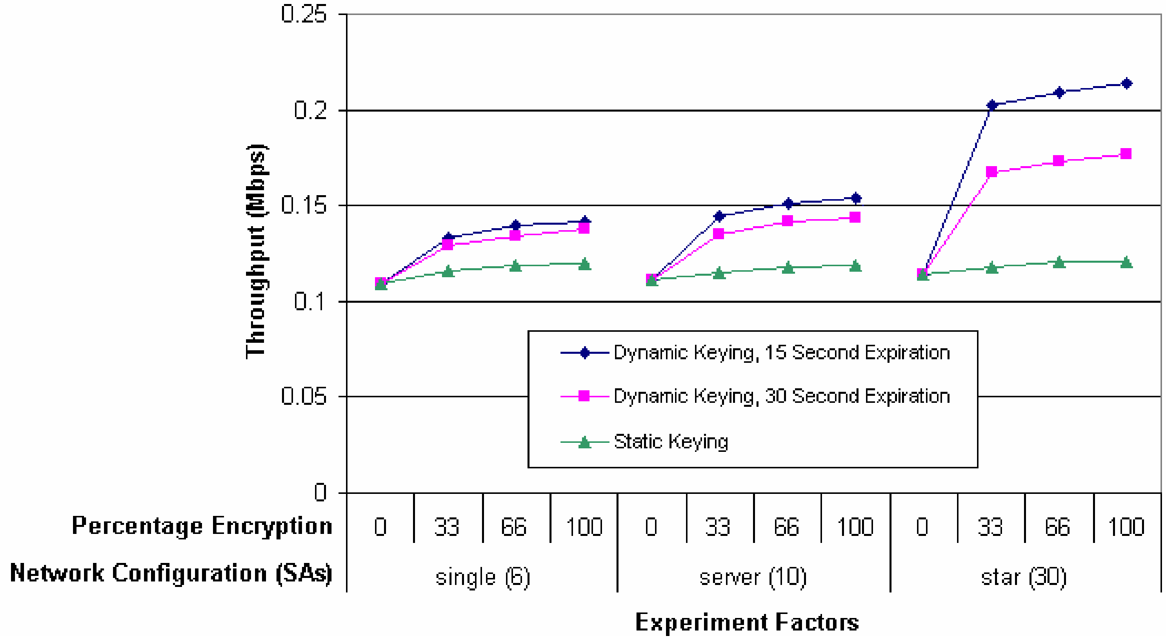


Figure 4.2: Mean Throughput for Experiments with Mixed Attack Payloads

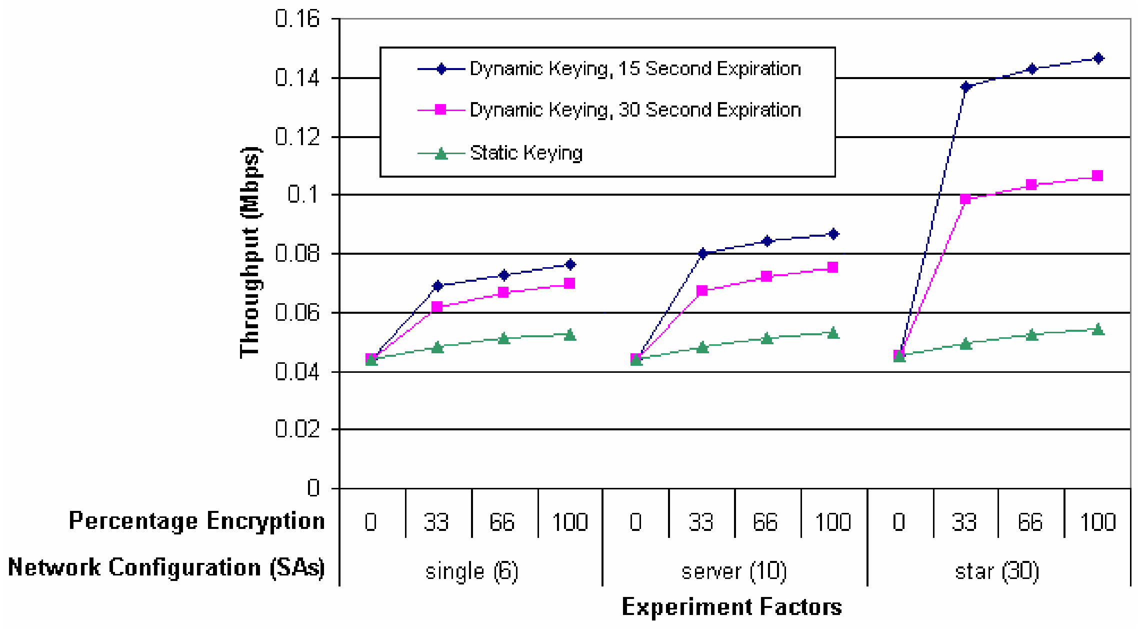


Figure 4.3: Mean Throughput for Experiments with Small Attack Payloads

CPU is not even close to being overloaded while packets are being dropped discounts the possibility the decryption/IDS loading was too much for the CPU to handle. It could be due to the multi-threaded operation of the IDS, contrasted to Snort's single-threaded operation, which, as an open-source IDS standard, is capable of handling a much higher throughput than was presented in these experiments. With several threads vying for control of the CPU, it is possible that packets are dropped while the packet-handling thread is not in control. Some Internet references indicate that the default value for BPF_BUFSIZE of 4KB is too small for fast packet sniffing applications. External empirical trials estimate that the buffer size should be set as high as 4MB for sniffing applications [Bro04, Fre04]. However, review of the Snort source code shows that Snort does not directly modify the value, rather that it uses the value overridden by *libpcap* of not more than 32KB (in pcap-bpf.c, the BIOCGBLEN ioctl is used to set the read buffer size) [Pca04, Roe04, Sou03]. Since *server* also uses *libpcap*, it should not be affected any differently than Snort. Regardless of *why* the drop rates are high with higher throughputs, the effect is eliminated for the experiments by studying the response to workloads at (low) throughputs which make drop rates insignificant.

Observing Figures 4.1-4.3 shows the obvious trend that a higher percentage of encrypted traffic and configurations requiring more security associations (SAs) yield higher throughputs. Table 4.1 summarizes t-tests which are performed on every pair of experiments with throughput as the response variable. For each factor, the percentage of statistically different experiments is determined by isolating the group of experiment pairs which vary only in that factor, and determining how many of those total comparisons show a statistical difference with 95% confidence.

Table 4.1 Percentages of Comparisons Showing Statistical Differences with 95% Confidence, Specific to Variation in a Particular Factor

Factor	Total Number of Comparisons	Number of Comparisons Showing Differences	Percentage of Comparisons Showing Differences	95% Confidence Interval for Percentage of Comparisons Showing Differences
Key Method	81	78	96.30%	92.18 – 100%
Configuration	90	73	81.11%	73.02 – 89.20%
% Encryption	108	97	89.81%	84.11 – 95.52%

For all attack payload sizes and percentages of encrypted traffic, there is a statistically significant difference in throughput between single/star and server/star configurations with 95% confidence. In the majority of cases, however, there is not a significant difference between single/server configurations, as there is less of a change between number of required SAs and therefore a smaller difference in key negotiation traffic. Recall that single configuration requires six SAs and server mode requires 10, while star mode requires 30.   Differences in key method greatly affect the throughput, and a statistical difference is shown in nearly every comparison. While percentage encryption does not have as large of an impact on throughput, almost 90% of comparisons do show a statistically significant difference when only this factor is varied.

The trend of higher throughput for higher encryption percentage and more SAs is logical since the majority of the goodput, in the form of attack payloads, remains the same across all variations of factors except payload size. As IPSec is applied, the ESP header and tail are added to encrypted payloads as well as padding, which can be up to 63 bytes with TripleDES due to the 64 byte block size. Note that HMAC-SHA1 authentication is added to the ESP packets in the dynamic-keyed experiments, though this addition is insignificant, accounting for less than 1% of total throughput. The more significant differences in throughput are due to the number of SAs required, which generate quite a bit of traffic between hosts for IKE and IPSec key negotiation. Figures 4.4-4.6 show a ratio of mean goodput to mean throughput for each payload size.

As the figures show, the ratio of good data transmitted to the total data transmitted drops anywhere between 0.06 to 0.16 from single mode to star mode, as more SAs are added increasing the network overhead. The surprising result, seen on Figure 4.6, is that with small payloads, 100% encrypted traffic, and star configuration, only about 6.5% of total network traffic is user data. This can have a significant effect on the IDS, which scans all traffic. In these experiments, UDP traffic is not analyzed by the IDS, simply logged and discarded. However, since IKE and IPSec exchanges take place using the

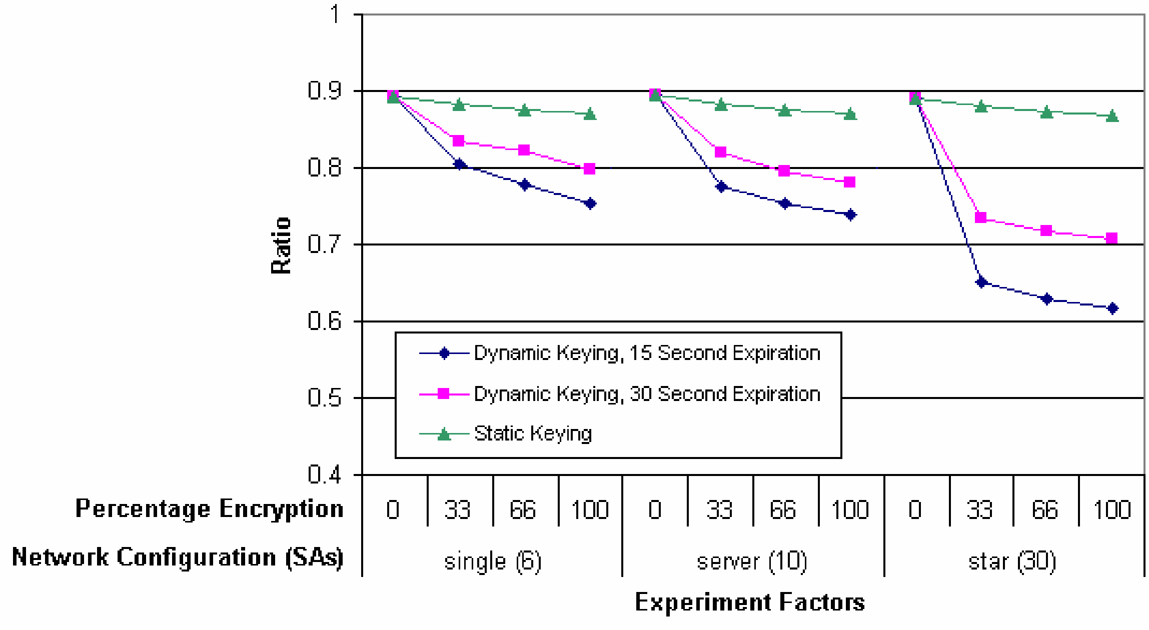


Figure 4.4: Ratio of Mean Goodput to Mean Throughput for Experiments with Large Attack Payloads

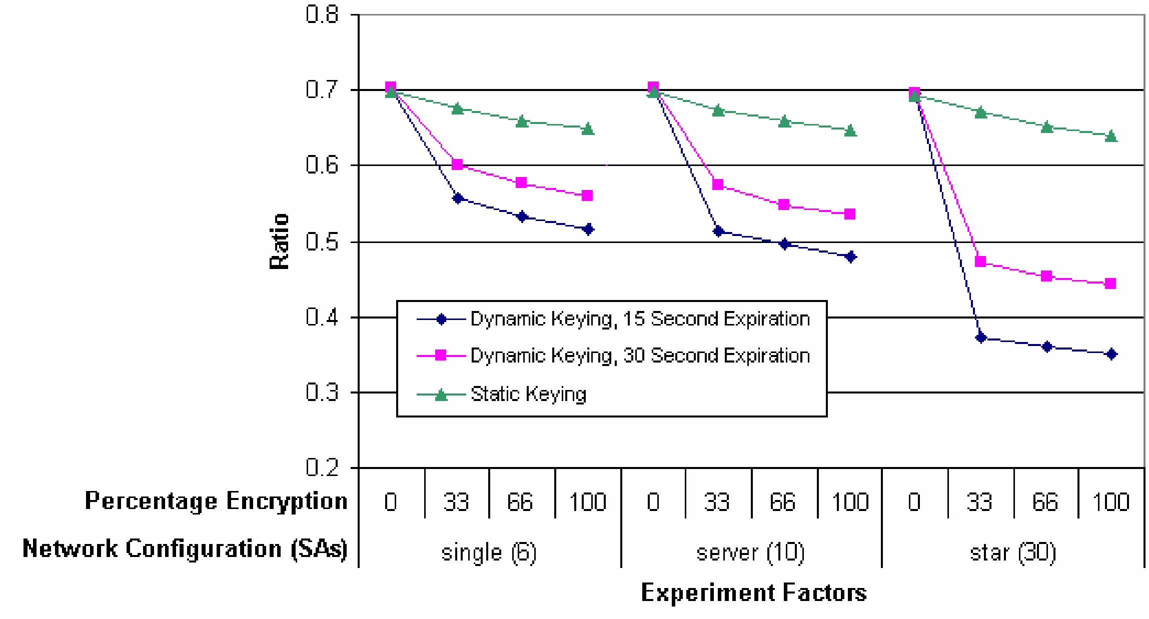


Figure 4.5: Ratio of Mean Goodput to Mean Throughput for Experiments with Mixed Attack Payloads

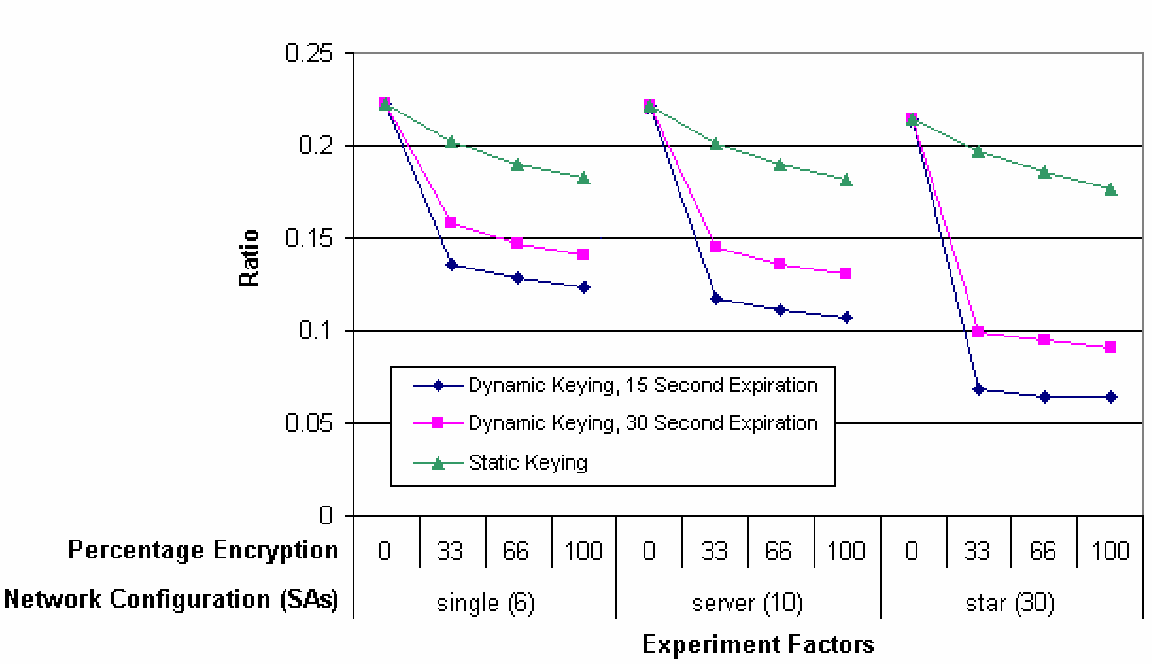


Figure 4.6: Ratio of Mean Goodput to Mean Throughput for Experiments with Small Attack Payloads

UDP protocol under the *racoon* implementation, a realistic IDS which scans UDP packets has to contend with a significant amount of traffic brought on by the IKE/secret-key sharing framework to ensure complete detection capability. While a typical TCP attack, such as a buffer overflow in clear text, may generate only a small amount of TCP traffic, the ensuing IKE and IPSec key negotiations from the same attack over an encrypted channel can create quite a bit of added traffic for the IDS to examine.

Analysis of variance (ANOVA) on the throughput response is shown in Table 4.2, grouped by attack payload size. The F-Ratios indicate all factors and interactions are significant with 95% confidence. The first-order factors explain the highest amount of variation, though no one factor stands out across all three payload sizes. For large attack

Table 4.2: Analysis of Variance for Throughput

Large Attack Payloads

Factors	DF	SumOfSq	MeanSq	F-Ratio	F-Table	P Value	% Variation
km	2	0.065825	0.032913	190.0559	3.01	0	17.35%
pe	3	0.11665	0.038883	224.5331	2.62	0	30.74%
nc	2	0.070623	0.035312	203.909	3.01	0	18.61%
km * nc	4	0.027324	0.006831	39.44572	2.39	0	7.20%
km * pe	6	0.023238	0.003873	22.36456	2.12	0	6.12%
nc * pe	6	0.010305	0.001717	9.917445	2.12	4.72E-10	2.72%
km * nc * pe	12	0.00935	0.000779	4.499212	1.77	1.05E-06	2.46%
Error	324	0.056108	0.000173				14.79%
Total	359	0.379423					

Mixed Attack Payloads

Factors	DF	SumOfSq	MeanSq	F-Ratio	F-Table	P Value	% Variation
km	2	0.077932	0.038966	622.9065	3.01	0	25.10%
pe	3	0.075131	0.025044	400.3445	2.62	0	24.19%
nc	2	0.056984	0.028492	455.4659	3.01	0	18.35%
km * nc	4	0.030197	0.007549	120.6822	2.39	0	9.72%
km * pe	6	0.026341	0.00439	70.18098	2.12	0	8.48%
nc * pe	6	0.013584	0.002264	36.19184	2.12	0	4.37%
km * nc * pe	12	0.010097	0.000841	13.45139	1.77	0	3.25%
Error	324	0.020268	6.26E-05				6.53%
Total	359	0.310534					

Small Attack Payloads

Factors	DF	SumOfSq	MeanSq	F-Ratio	F-Table	P Value	% Variation
km	2	0.079815	0.039908	4598.354	3.01	0	27.58%
pe	3	0.073147	0.024382	2809.447	2.62	0	25.27%
nc	2	0.051858	0.025929	2987.686	3.01	0	17.92%
km * nc	4	0.029816	0.007454	858.8884	2.39	0	10.30%
km * pe	6	0.026709	0.004451	512.9192	2.12	0	9.23%
nc * pe	6	0.015305	0.002551	293.9161	2.12	0	5.29%
km * nc * pe	12	0.009962	0.00083	95.65455	1.77	0	3.44%
Error	324	0.002812	8.68E-06				0.97%
Total	359	0.289424					

Legend

Symbol	Factor
km	Key Method
pe	Percentage of Encrypted Traffic
Nc	Network Configuration

payloads, percentage of encrypted traffic affects throughput more than in the cases of small and mixed attack payloads. This can be attributed to the higher overall throughput present in large-payload scenarios—the key method and network configuration add a smaller relative overhead to the throughput, whereas percentage of encrypted traffic adds more overhead on a more consistent per-packet basis.

4.4 Attack Detection Rates, Dropped Packets, and Undecryptable Packets

One of the goals for the experiments was to have a high, if not perfect, attack detection rate given that an intrusion detection system which does not provide thorough coverage is not worth much. Average detection rates for large, mixed, and small payload sizes are 98.92%, 95.85%, and 98.3% respectively, across all experiments for that payload size. The actual detection rates range from a low of 91.75% to a high of 100.83%, with 89% of the experiments yielding a detection rate of 95% or better. Low detection rates are generally correlated to dropped packets, and packets which could not be decrypted, an event discussed later in the section. Detection rates over 100% do occur on a few occasions (16% of experiments) and are due to TCP retransmits by the attacking hosts. In some cases the receiving host fails to ACK a particular packet, and the packet is resent. As the retransmits occur at the TCP layer and not the application later, they are not recorded as additional attacks sent—however the IDS intercepts and alerts on the attacks all the same, driving the attack detection rate slightly over 100%. The uncertainty of the TCP retransmits must be considered in the lower bound of the attack detection rate as well. Experiments showing no packet loss and independent trials conducted after data collection indicated that up to 2.5% of attacks were duplicated in a TCP retransmit, though a bound is not guaranteed. TCP retransmits were unexpected, and were not tracked for the experiments. Therefore they introduce some error into the attack detection rate which is not fully accounted for. Table 4.3 summarizes attack detection rate and other statistics discussed in this section.

Packet drop statistics are recorded by pcap, and as mentioned in Section 4.3 are necessarily low. With the exception of one experiment (number 56 from Appendix A), which yields a mean drop rate of 5.15%, all drop rates are below 5%. Experiments with mixed attack payloads tended to drop more packets than the other sizes, due to the fact the attacks were being sourced at the same rate as small attack payloads but with a higher average size (see Table 3.3). That, the 100% encrypted traffic, and the 15 second key expiration time for experiment 56 are all factors influencing the slightly higher drop rate. The overall mean drop rates for large, mix, and small payloads are 0.55%, 2.59%, and 1.51% respectively and don't follow any particular trend across factors.

Table 4.3: Attack Detection Related Statistics

Statistic	Attack Payload Size	Low Value	Mean	High Value
Attack Detection Rate	Large	94.89%	98.92%	100.83%
	Mix	91.75%	95.85%	100.04%
	Small	95.85%	98.30%	100.04%
Packet Drop Rate	Large	0.00%	0.55%	3.20%
	Mix	0.72%	2.59%	5.15%
	Small	0.33%	1.51%	4.55%
Undecryptable Packets (raw)	Large	0.00	1.17	5.80
	Mix	0.00	3.87	75.50
	Small	0.00	1.28	6.40

Finally, *server* keeps a record of packets which it is unable to decrypt. Generally, the number of undecryptable packets is quite low, sometimes zero or around 5 in 20,000+ packets examined. The only way this occurs is if the replicated SAD does not contain the right decryption key, in which case decryption cannot be completed. Some transient behavior is observed in which a host negotiates an IPSec SA and transmits an attack before the key has been entered into the IDS' database, in which case *server* may be unable to decrypt a handful of packets before the key is present. However, in one experimental run, a very high number of packets were undecryptable. Specifically, in run number 7 of experiment 47 (see Appendix A) 564 packets were undecryptable. This is almost certainly due to the *server* software missing a transmitted key, or *racoon2* failing to transmit the key to *server*. The *server* program is a passive listener for keys and does not attempt to actively obtain a decryption key from a host on the network, rather it relies on the hosts to send updated keys proactively. Therefore, if a key transmission somehow times out or fails, *server* will not have the means to decrypt any packets until a new key is generated and updated.

Thus this emphasizes the tradeoff between complete protection and non-interference. A design decision is made not to provide *server* a key solicitation capability due to the possibility it would further load down the network and hosts. If key solicitation were allowed, perhaps no packets would be "undecryptable." However, the penalty on IDS performance as decryption keys are retrieved from a host across a network can be severe.

4.5 IDS Related Traffic

In each experiment, the packet log is analyzed to determine the percentage of traffic entering or leaving the IDS host. Figures 4.7-4.9 display the results. Since IDS related traffic in static keying mode is either zero or negligible, it is not displayed in the figures.

The graphs show that in some cases a very high percentage, up to 58.6%, of traffic is IDS related. What they don't show, however, is that the raw throughput due to the IDS and secret-key sharing framework is very consistent for any keying mode/network configuration combination across other factors. This is logical, since key refreshes occur at fixed time intervals (15 seconds, 30 seconds, or never) and for a specific number of SAs for each configuration. Rather, the amount of non-IDS related throughput changes and thus affects the percentages shown.

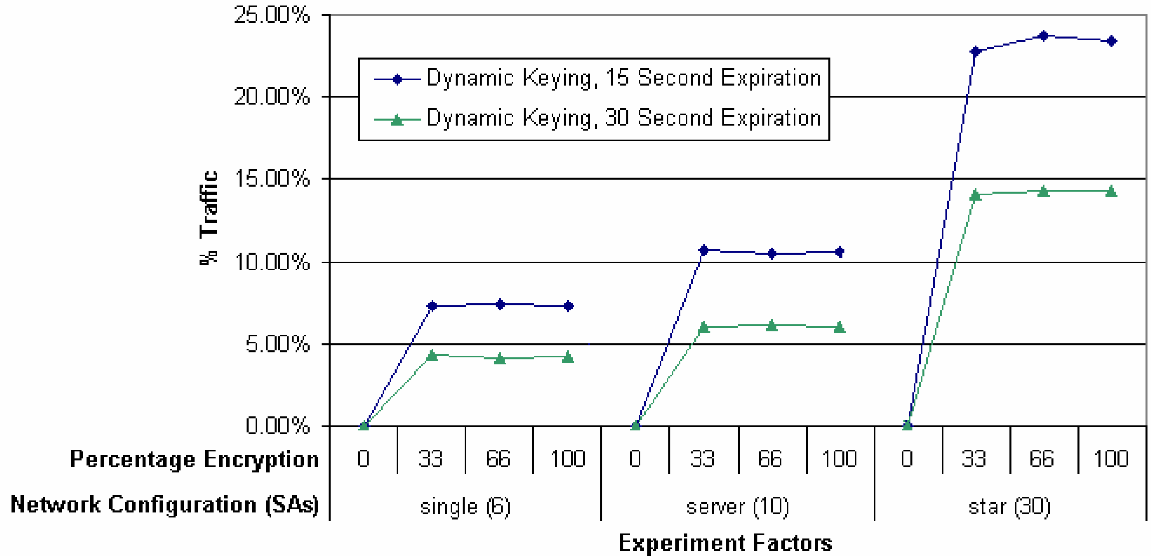


Figure 4.7: Percentage of IDS Related Traffic for Experiments with Large Attack Payloads

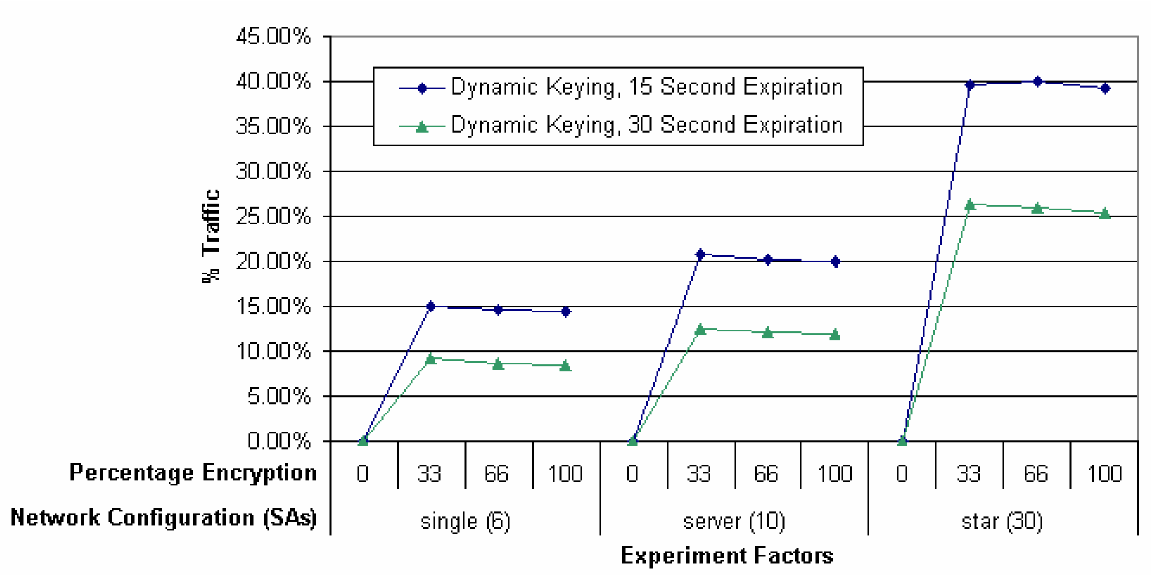


Figure 4.8: Percentage of IDS Related Traffic for Experiments with Mixed Attack Payloads

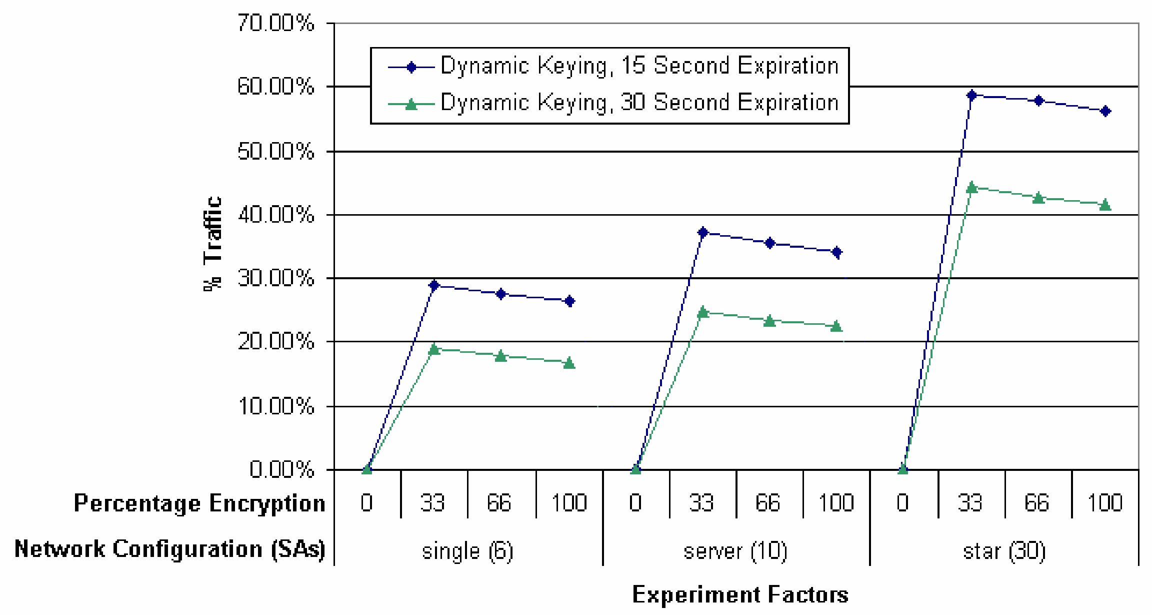


Figure 4.9: Percentage of IDS Related Traffic for Experiments with Small Attack Payloads

The secret-key sharing framework does generate a fair amount of traffic. The actual SAD updates are quite small, however they are transmitted over IPSec-secured channels, which require the standard fare of IKE and IPSec key negotiation traffic. This is the simplest method of ensuring the encryption keys are afforded the same level of protection as the data they are meant to protect. However, this method clearly can create a substantial amount of overhead related to how many hosts are on the network. As offered load increases between hosts, though, the percentage should drop as SA updates are time dependent and not throughput dependent. Note that if only one IPSec endpoint resides on the network local to the IDS, the traffic is cut in half as the remote node will not be transmitting SA updates to the IDS. Additionally, this traffic could be decreased by using the UDP protocol to transmit SA updates to the IDS, or by protecting only host-to-IDS traffic with IPSec, rather than host-to-IDS and IDS-to-host since the return traffic does not contain any sensitive data.

4.6 IDS CPU Utilization

A primary research focus is to determine what, if any, inferences about the secret-key sharing framework can be drawn by observing the IDS CPU utilization across variation of factors. Figures 4.10-4.12 show the mean IDS CPU utilization for the experimental factors. Depending upon experimental factors, using dynamic keying with 30 second key expiration times increased CPU load by up to 14.5% (relative to static-keyed experiments), and dynamic keying with 15 second key expiration time increased the load by up to 20.7%, as determined with the following equation.

$$\%\ \mathit{Increase} = 100 * \frac{\mathit{CPU\ Utilization\ with\ Dynamic\ Keying} - \mathit{CPU\ Utilization\ with\ Static\ Keying}}{\mathit{CPU\ Utilization\ with\ Static\ Keying}}$$

For the most part, the results confirm what should be intuitive. As encryption percentage is increased, CPU utilization increases. In a few cases, this does not strictly hold, but the trend is evident. Similarly, as the number of SAs in circulation increases (expiration times and configurations change), the CPU utilization is slightly higher. In 13 of 30 cases, the mean CPU utilization actually decreased from single configuration to server. Seeing that server configuration requires more SAs than single, this result is unexpected. However, there is not a statistical difference at the 95% confidence level, so that result is more or less inconclusive.

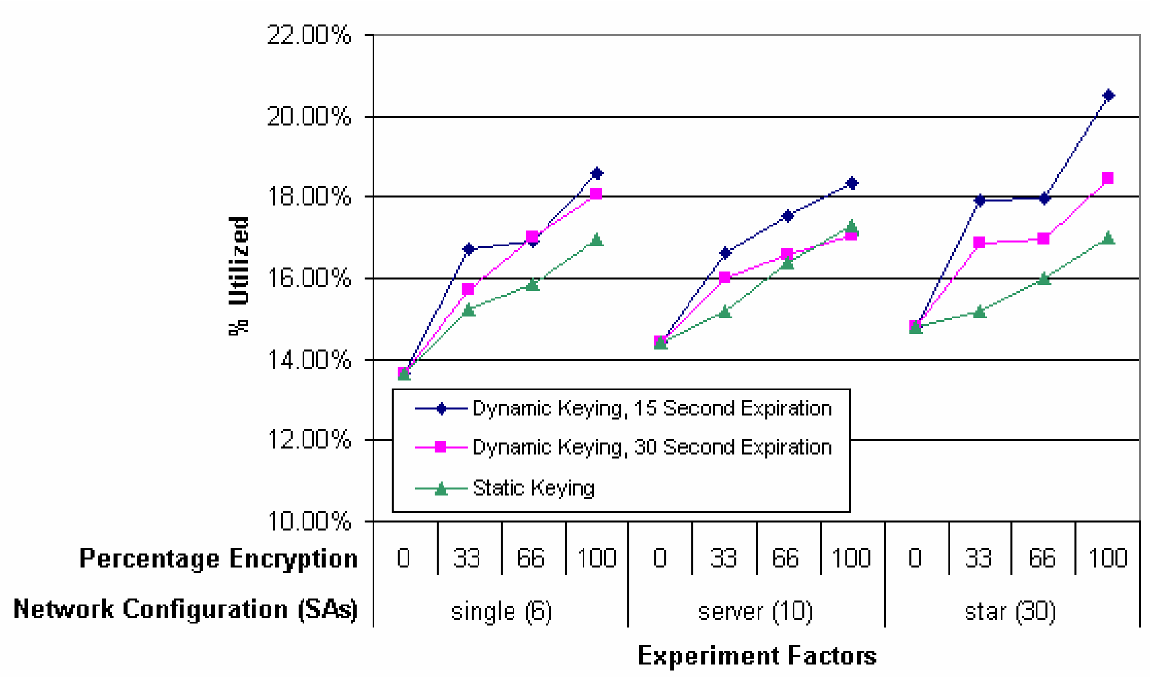


Figure 4.10: Mean IDS CPU Utilization for Experiments with Large Attack Payloads

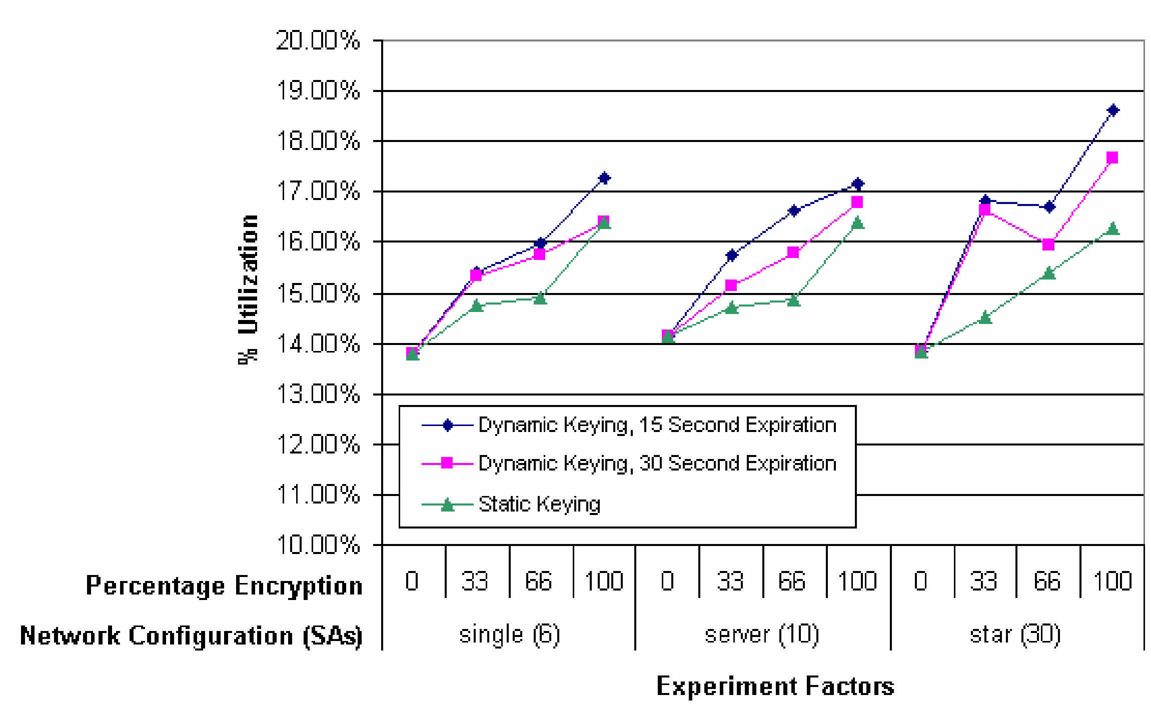


Figure 4.11: Mean IDS CPU Utilization for Experiments with Mixed Attack Payloads

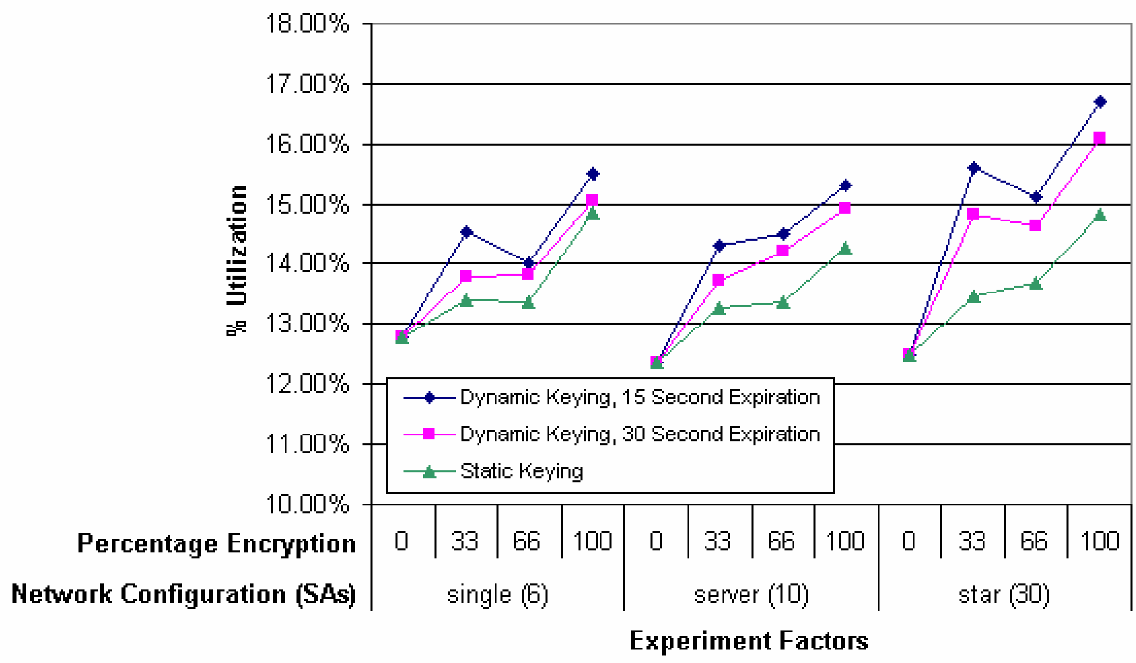


Figure 4.12: Mean IDS CPU Utilization for Experiments with Small Attack Payloads

Observing experiments with 0% encryption shows no real trend applying to the CPU utilization across network configurations when using static keying. This indicates that performance differences due to configuration changes in the dynamic-keyed experiments are primarily due to the key exchanges themselves, and not related to server's operations of retrieving keys from and maintaining the SAD.

In most cases, experiments using dynamic keying with 30 second key expiration times do not incur enough of a penalty on the CPU to statistically differentiate them from static-keyed experiments with the same workload characteristics with 95% confidence. However, in all but three cases there is a statistically significant difference between static-keyed experiments and dynamic-keyed experiments with 15 second expiration times. Figures 4.13-4.15 show these measurements with their confidence intervals. On overlapping confidence intervals, the t-test confirms the statistical difference in all cases except three. Those inconsistencies are most likely explained by experimental error.

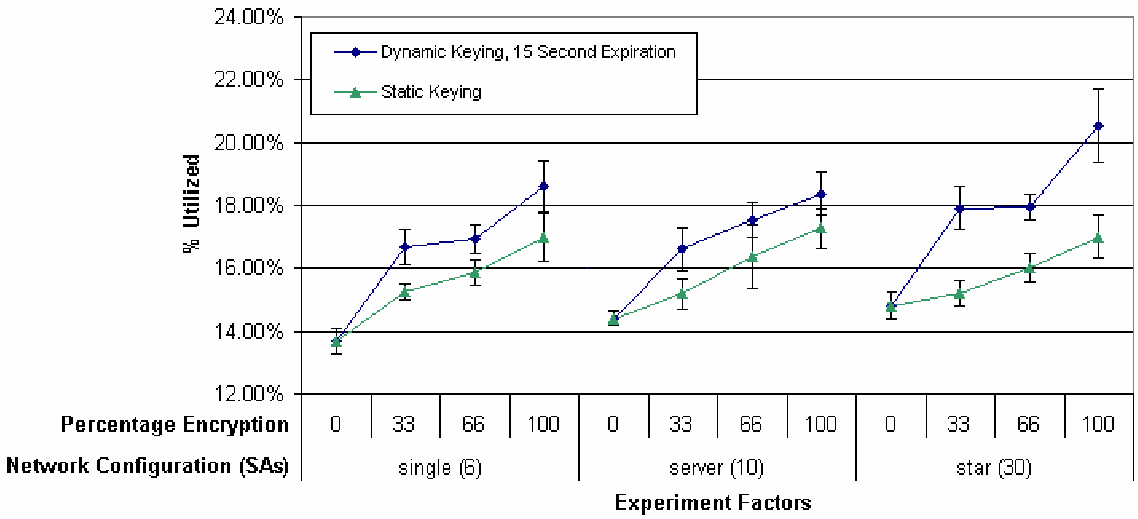


Figure 4.13: Mean IDS CPU Utilization for Experiments with Large Attack Payloads, Including 95% Confidence Interval

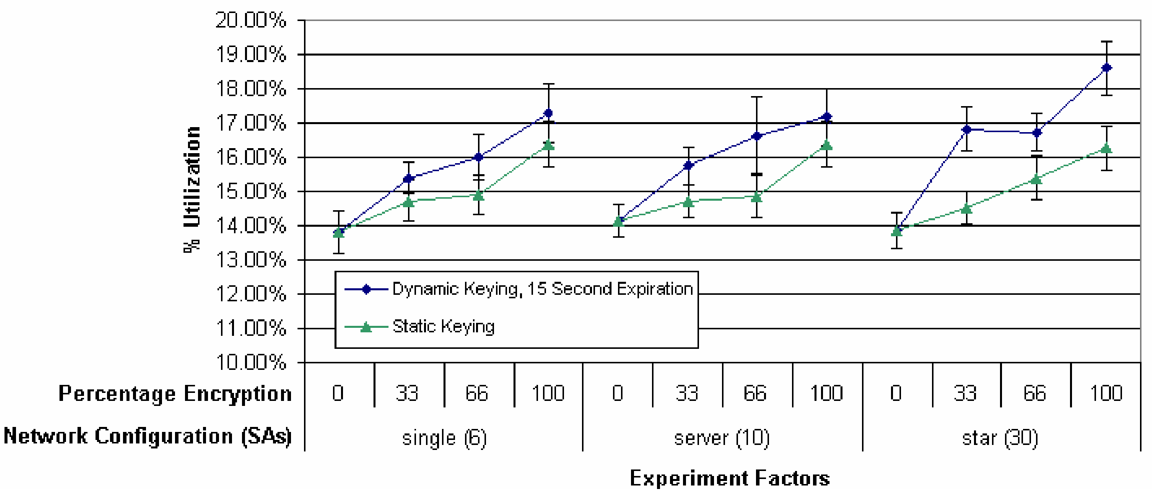


Figure 4.14: Mean IDS CPU Utilization for Experiments with Mixed Attack Payloads, Including 95% Confidence Interval

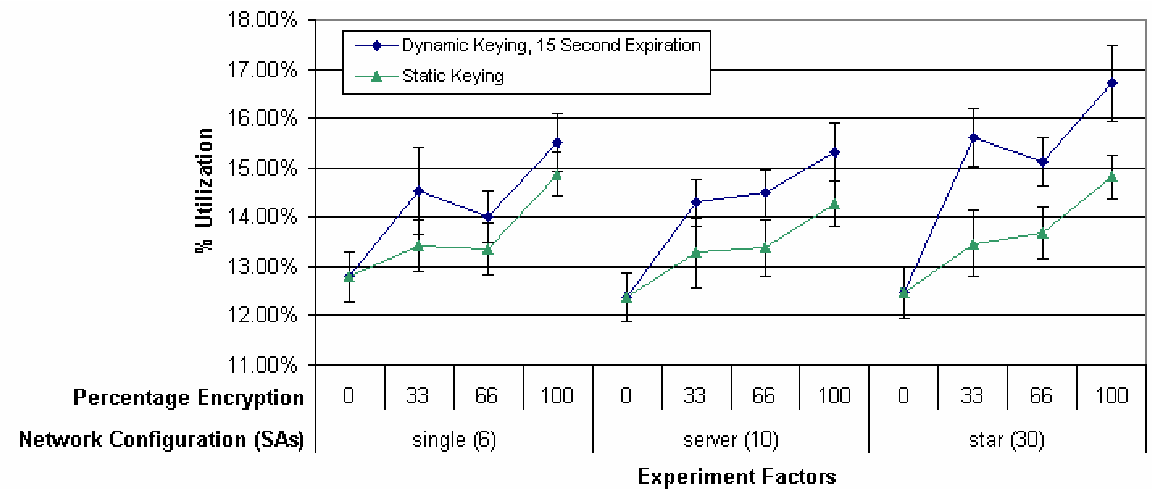


Figure 4.15: Mean IDS CPU Utilization for Experiments with Small Attack Payloads, Including 95% Confidence Interval

ANOVA was performed on the CPU utilization metric using Mathematica. Table 4.4 shows a summary of the allocation of variation for each attack payload size. The first thing to note is the percentage of variation attributed to experimental error. There are a number of possible explanations for this error. The first is simply the unpredictable nature of the distributed system and Ethernet network. Although every effort is made to control the workload, the manner in which it is supplied (at the application layer) leaves much underlying activity to the whim of the operating system. For example, the TCP retransmissions discussed in Section 4.5 have a negative effect on the IDS CPU utilization, as they artificially increase the workload—and those retransmissions, unfortunately, are seemingly unpredictable. On top of trusting the FreeBSD operating system and all host-to-host interactions on the Ethernet LAN to act in a perfectly uniform manner across experiments, the impact of VMWare must also be considered. Although the FreeBSD guest operating system is installed on top of a clean-install Windows 2000 host, with no extraneous programs running in the background, Windows 2000 is certainly undergoing a minimal level of activity in the background. Since that activity is ignored in these experiments, it is likely to introduce some variation which must be attributed to experimental error. According to the F-ratios shown in Table 4.4, each factor is significant with 95% confidence except for the third order interaction of key method, configuration, and percent encryption using small and mixed sized payloads. Not surprisingly, the largest source of variation in CPU utilization is due to the percentage of encrypted traffic present in the workload. TripleDES decryption is a cycle-intensive operation, and without

Table 4.4: Analysis of Variance for IDS CPU Utilization

Large Attack Payloads

Factors	DF	SumOfSq	MeanSq	F-Ratio	F-Table	P Value	% Variation
km	2	0.010648	0.005324	77.76557	3.01	0	9.56%
pe	3	0.065963	0.021988	321.1613	2.62	0	59.20%
nc	2	0.003907	0.001953	28.53146	3.01	3.86E-12	3.51%
km * nc	4	0.001791	0.000448	6.538755	2.39	4.52E-05	1.61%
km * pe	6	0.004075	0.000679	9.919724	2.12	4.70E-10	3.66%
nc * pe	6	0.001393	0.000232	3.391675	2.12	0.002936	1.25%
km * nc * pe	12	0.001458	0.000122	1.775122	1.77	0.051176	1.31%
Error	324	0.022182	6.85E-05				19.91%
Total	359	0.111416					

Mixed Attack Payloads

Factors	DF	SumOfSq	MeanSq	F-Ratio	F-Table	P Value	% Variation
km	2	0.006224	0.003112	40.93368	3.01	1.11E-16	7.64%
pe	3	0.042969	0.014323	188.3878	2.62	0	52.76%
nc	2	0.001875	0.000937	12.33028	3.01	6.90E-06	2.30%
km * nc	4	0.000858	0.000214	2.820622	2.39	0.025199	1.05%
km * pe	6	0.002281	0.00038	5.000203	2.12	6.44E-05	2.80%
nc * pe	6	0.001273	0.000212	2.789802	2.12	0.011665	1.56%
km * nc * pe	12	0.001332	0.000111	1.46006	1.77	0.137788	1.64%
Error	324	0.024633	7.6E-05				30.25%
Total	359	0.081445					

Small Attack Payloads

Factors	DF	SumOfSq	MeanSq	F-Ratio	F-Table	P Value	% Variation
km	2	0.005161	0.00258	41.271	3.01	1.11E-16	7.76%
pe	3	0.033955	0.011318	181.035	2.62	0	51.06%
nc	2	0.0025	0.00125	19.99367	3.01	6.49E-09	3.76%
km * nc	4	0.000729	0.000182	2.914574	2.39	0.021588	1.10%
km * pe	6	0.001855	0.000309	4.943752	2.12	7.38E-05	2.79%
nc * pe	6	0.00165	0.000275	4.397779	2.12	0.000273	2.48%
km * nc * pe	12	0.000392	3.27E-05	0.522266	1.77	0.900064	0.59%
Error	324	0.020257	6.25E-05				30.46%
Total	359	0.066498					

Legend

Symbol	Factor
km	Key Method
pe	Percentage of Encrypted Traffic
nc	Network Configuration

dedicated cryptographic hardware the burden on the CPU increases greatly as more encrypted packets are encountered. After experimental error, key method is responsible for the next largest source of variation, however it is only roughly 1/6th as significant as the percent encryption. Network configuration explains a surprisingly small amount of variation, as apparently the number of valid SAs in the system does not in itself generate much overhead on the IDS CPU. The interacting effects of key method and percent encryption cause some variation, due to the fact that percent encryption directly affects access to the SAD, whose maintenance is dictated by the key method. Finally, the interactions of key method with configuration and configuration with percent encryption are responsible for a small amount of variation.

Though these results cannot be scaled to all situations, they indicate that IDS CPU performance does not depend significantly on the number of SAs in the database nor the number of SA updates it is receiving through the secret-key sharing framework. Although key method is not as significant as percentage of encrypted traffic, any parties wishing to use this type of secret-key sharing system need to consider the keying method carefully especially if using dynamic keying, since as key expiration time increases it becomes more and more like static keying and the difference in IDS performance will be negligible. If, however, very short key expiration times are used and many SAs are generated, the IDS performance will suffer increasingly. Drawing from Table 4.4 and the notions presented in Section 4.3 regarding the goodput to throughput ratio, it seems likely that increased CPU utilization correlated to key method is due more to the increased key negotiation traffic which the IDS must process than the number of SAs actually in circulation.

As a final note, it is important to realize that much of the key negotiation traffic is due to IKE and IPSec exchanges between hosts, and not directly related to the IDS computer. Therefore, adding IDS functionality with the secret-key sharing framework to a network *already* using IPSec protection with dynamic keying creates only a fraction of the overhead that it would first appear. The large difference is when the secret-key sharing framework is added to a network protected only by static keying, in which case no key negotiation is required at all. Bringing this into context shows that adding an IDS to an IPSec protected network with dynamic keying may be a very viable option.

4.7 Limitations

This section explores research limitations, and how the experiments could be improved.

<u>4.7.1 VMWare</u> The use of VMWare eased the setup and execution of the experiments. However, judging from the high experimental errors this may not have been the best choice. There are clearly factors influencing the experiments which are not accounted for, and one such factor likely is the influence of VMWare and the underlying host operating system, Windows 2000. If each host was running FreeBSD as a native operating system, then the potential influence would be eliminated.

<u>4.7.2 Throughput</u> Although these experiments provide useful trend data, it is clear that the framework is not ready for any sort of real world use. Admittedly, as

designed the framework was not intended for real world use, however it would be beneficial to be able to subject it to more demanding workloads for further trend analysis. One of the more interesting future options is to insert the IDS and secret-key sharing framework on a larger network with more SA activity, however this would be all but impossible until the packet dropping issue is resolved.

4.8 Summary

This chapter discussed the results of the experiments described in Chapter 3, comparing the results of IDS efforts in static-keyed encrypted networks with those in dynamic-keyed networks using the developed secret-key sharing framework. Though use of the secret-key sharing framework incurred up to a 20.7% CPU utilization penalty on the IDS computer, the improved security of dynamic keying over static keying makes secret-key sharing for IDS an option worth pursuing.

5 Conclusions

5.1 Research Contribution

This research, driven by the advent of IPv6, explores a difficult problem exacerbated by IPv6's inherent security attributes. A secret-key sharing framework was developed utilizing new and existing code to enable intrusion detection in an IPv6 enabled network employing IPSec protection. This framework provides a basis for future research and/or implementation of such a system. Additionally, the experimental results herein provide insight into the traffic patterns of a LAN using the framework, and the added burden placed on the IDS host which is already a highly taxed resource in most networks.

As a byproduct of the primary research goals, a pair of traffic generation programs, *attacknet* and *attackclient*, were created. These programs provide a method for testing intrusion detection systems from the application layer for up to six network hosts. The advantage of this type of traffic generation is it allows standard kernel processing including IPSec. The programs can easily be modified to include any number of hosts, to permit more strenuous testing.

5.2 Conclusions

Experimental results show that a secret-key sharing framework for protecting IPSec traffic is a viable option. This is not to say that the particular framework proposed here, or even the methods employed by the framework, are the "best" options. Rather, despite identified shortcomings of the proposed framework, the relative overhead imparted on the network and IDS CPU is not overwhelming. The door on such a secret-key sharing option over static keying, at the very least, is not summarily closed.

To review, the first research goal was to develop the proposed key sharing framework and that goal was clearly accomplished. Though the workloads presented in these experiments were not strenuous by today's standards, the framework effectively enabled intrusion detection on encrypted traffic with a high level of accuracy.

The second research goal was to provide different workloads to the system and characterize traffic patterns. Analysis from Chapter 4 indicates that in some circumstances the framework is responsible for a high percentage of total network throughput (i.e., star-configured scenarios using small attack payloads, 100% encryption and dynamic keying with 15 second expiration times showed that nearly 60% of network traffic was directed to or from the IDS host). That is certainly not desirable performance for a "low-overhead" framework, though it does provide better performance on more average workloads such as mixed payloads in the single or server configurations using 30 second key expiration times. In these scenarios, only 10-15% of traffic was IDS related.

The final research goal was to examine the CPU utilization of the IDS host, with the hypothesis that the packet decryption would be the performance bottleneck. For the scenarios considered, the hypothesis holds. However, the data also indicates that as LAN size increases (i.e., the number of SAs being generated over a period of time increases) this may not always be case. The key method was shown to be responsible for a significant 7-9% of variation in CPU response, leading to the conclusion that as key traffic steadily increases, the load on the CPU will likewise. While it's true that the impact of key method was not as significant as that of the percentage of encrypted traffic, if the cryptographic operations were offloaded to a coprocessor (a likely event for a high-throughput network), changes in key method (SA creation rate) becomes the highest source of variation in CPU load.

5.3 Future Research Ideas

There is much additional research which can be done in this area. Follow-on research could be performed in some of the following areas.

5.3.1 Key solicitation As mentioned previously, the *server* was not given the capability to request keys from hosts on the network, and discarded any packets which it was unable to decrypt immediately. It would be possible for the *server* to be reactive to encountering new SPIs, where it would request new keys from the hosts and cache them locally. This may not add much of a penalty to the performance of the framework, and at least would guarantee complete protection.

5.3.2 Higher Throughput and More Hosts If any future work is done with this specific framework, the packet dropping issue must be resolved. Certainly at this point, the framework is a nice idea but hardly qualifies as useful due to the low data rates it can effectively protect. With that problem solved, the framework could be integrated into a LAN with more hosts to determine the effects of the increased host count and SA activity.

5.3.3 Include Different Operating Systems At the time of this research effort, the only operating system supporting both IPv6 and IPSec concurrently was FreeBSD and variants. In the near future, these capabilities will have to be integrated into other operating systems such as Windows in order for consumers to get the full benefit of IPv6. At that point, the framework could be ported to these new platforms and performance evaluated.

5.3.4 Involve a Cryptographic Coprocessor With cryptographic functions off-loaded to a coprocessor, this would leave the CPU on the IDS host free to deal solely with key reception (or solicitation) and intrusion detection functions. Many coprocessor options are available currently which support data encryption/authentication rates up to 1.0Gbps for IPSec processing [Saf04, Sun04]. This alone could improve the system as a whole, and make the secret-key sharing idea a more realistic option as decryption proved to be the bottleneck for these experiments.

5.3.5 Possible Improvements to *server*, *racoon2*, and the General Framework Certain design decisions were made which were not necessarily the best or most efficient options. For example, when keys were exchanged with the IDS, both channels of communication were protected with IPSec (both to and from the IDS). In reality, there is no secret information transferred from the IDS to a host, so that channel could be left in the clear somewhat lessening communication burden on the IDS. Additionally, taking the original *racoon* implementation of IKE and IPSec as inspiration, the secret-key sharing traffic could have been accomplished over UDP rather than TCP and possibly made more efficient.

5.3.6 Real-World IDS Integration The custom IDS developed for these experiments is not a good representation of the capabilities generally available to the public. Commercial IDS' and the open-source Snort include features such as stream assembly and complex rulesets which make them much more effective than a simple per-packet pattern matching engine. There is no particular reason why the framework couldn't be integrated into Snort. The release available at the time of the research did not support IPv6 traffic other than recognizing it.

Appendix A: Experiment Listing

Experiment Number	Network Configuration	Percentage of Encrypted Traffic	Key Method	Key Expiration Time (sec)	Attack Payload Size
1	star	33	static	n/a	small
2	star	33	static	n/a	mix
3	star	33	static	n/a	large
4	star	66	static	n/a	small
5	star	66	static	n/a	mix
6	star	66	static	n/a	large
7	star	100	static	n/a	small
8	star	100	static	n/a	mix
9	star	100	static	n/a	large
10	star	33	dynamic	30	small
11	star	33	dynamic	30	mix
12	star	33	dynamic	30	large
13	star	33	dynamic	15	small
14	star	33	dynamic	15	mix
15	star	33	dynamic	15	large
16	star	66	dynamic	30	small
17	star	66	dynamic	30	mix
18	star	66	dynamic	30	large
19	star	66	dynamic	15	small
20	star	66	dynamic	15	mix
21	star	66	dynamic	15	large
22	star	100	dynamic	30	small
23	star	100	dynamic	30	mix
24	star	100	dynamic	30	large
25	star	100	dynamic	15	small
26	star	100	dynamic	15	mix
27	star	100	dynamic	15	large
28	star	0	n/a	n/a	small
29	star	0	n/a	n/a	mix
30	star	0	n/a	n/a	large

Experiment Number	Network Configuration	Percentage of Encrypted Traffic	Key Method	Key Expiration Time (sec)	Attack Payload Size
32	single	33	static	n/a	mix
33	single	33	static	n/a	large
34	single	66	static	n/a	small
35	single	66	static	n/a	mix
36	single	66	static	n/a	large
37	single	100	static	n/a	small
38	single	100	static	n/a	mix
39	single	100	static	n/a	large
40	single	33	dynamic	30	small
41	single	33	dynamic	30	mix
42	single	33	dynamic	30	large
43	single	33	dynamic	15	small
44	single	33	dynamic	15	mix
46	single	66	dynamic	30	small
47	single	66	dynamic	30	mix
48	single	66	dynamic	30	large
49	single	66	dynamic	15	small
50	single	66	dynamic	15	mix
51	single	66	dynamic	15	large
52	single	100	dynamic	30	small
53	single	100	dynamic	30	mix
54	single	100	dynamic	30	large
55	single	100	dynamic	15	small
56	single	100	dynamic	15	mix
57	single	100	dynamic	15	large
58	single	0	n/a	n/a	small
59	single	0	n/a	n/a	mix
60	single	0	n/a	n/a	large

Experiment Number	Network Configuration	Percentage of Encrypted Traffic	Key Method	Key Expiration Time (sec)	Attack Payload Size
61	server	33	static	n/a	small
62	server	33	static	n/a	mix
63	server	33	static	n/a	large
64	server	66	static	n/a	small
65	server	66	static	n/a	mix
66	server	66	static	n/a	large
67	server	100	static	n/a	small
68	server	100	static	n/a	mix
69	server	100	static	n/a	large
70	server	33	dynamic	30	small
71	server	33	dynamic	30	mix
72	server	33	dynamic	30	large
73	server	33	dynamic	15	small
74	server	33	dynamic	15	mix
75	server	33	dynamic	15	large
76	server	66	dynamic	30	small
77	server	66	dynamic	30	mix
78	server	66	dynamic	30	large
79	server	66	dynamic	15	small
80	server	66	dynamic	15	mix
81	server	66	dynamic	15	large
82	server	100	dynamic	30	small
83	server	100	dynamic	30	mix
84	server	100	dynamic	30	large
85	server	100	dynamic	15	small
86	server	100	dynamic	15	mix
87	server	100	dynamic	15	large
88	server	0	n/a	n/a	small
89	server	0	n/a	n/a	mix
90	server	0	n/a	n/a	large

Appendix B: Availability of Source Code and Configuration Files, and Data

Source code and configuration files for *server*, *racoon2*, *cpumon*, *attacknet*, and *attackclient* and all other files related to test bed setup are not included as part of this document. Additionally, all collected data is not printed in the document but is available.

Interested parties should direct their inquiries to:

Dr. Richard Raines
AFIT/ENG 2950 Hobson Way
Wright-Patterson AFB, OH
45433-7765

Bibliography

[Agi01] Agilent Technologies, “Mixed Packet Size Throughput,” 2001, http://advanced.comms.agilent.com/n2x/docs/insight/2001-8/TestingTips/1MxdPktSzThroughput.pdf

[Bal99] R. Baldwin, "Improving the Real-Time Performance of a Wireless Local Area Network," Ph.D. dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1999.

[Bro04] Bro Intrusion Detection System, 2004, http://www.bro-ids.org/

[CoM04] S. Convery, D. Miller, “IPv6 and IPv4 Threat Comparison and Best Practice Evaluation,” March 2004, http://www.seanconvery.com/v6-v4-threats.pdf

[DeH98] S. Deering and R. Hinden, “RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification,” IETF, December 1998, http://www.ietf.org/rfc/rfc2460.txt

[Eff04] Electronic Frontier Foundation, “Cracking DES,” 2004, http://www.eff.org/Privacy/Crypto_misc/DESCracker/

[Emi02] Jacqueline Emigh, “IPv6: What You Need to Know,” June 2002, http://www.networking.earthweb.com/netsp/article.php/1347761

[Eth04] *Ethereal – The World’s Most Popular Network Protocol Analyzer*, 2004, http://www.ethereal.com

[Fed04] Federal Financial Institutions Examination Council's Information Security Booklet, 2004, http://www.ffiec.gov/ffiecinfobase/booklets/information_secruity/04j_intrusion_detect%20_response.htm

[Fre04] FreeBSD-net Mail Archive, Conversation with Guy Helmer, 2004, http://lists.freebsd.org/pipermail/freebsd-net/2004-February/002863.html

[HaC98] D. Harkins and D. Carrel, “RFC 2409 – The Internet Key Exchange (IKE),” November 1998, http://www.ietf.org/rfc/rfc2409.txt

[HaM03] P. Harris and R. Mahmood, “A Case Study on Experimental IPv6 IDS,” presented at Hack in the Box Security Conference, Kuala Lumpur, Malaysia, 2003, http://www.scan-associates.net/papers/hitb2003-ids_ipv6.ppt

[Hei04] M. Heidari, “IPv6 Security Considerations,” September 2004, http://www.securitydocs.com/download.php?id=2545

[Hui96] C. Huitema, *IPv6, The New Internet Protocol*, New Jersey: Prentice Hall, 1996.

[Iss04] Internet Security Systems, 2004, http://www.iss.com/

[Jav04] Javvin Company, “IPv6 (IPng): Internet Protocol version 6”, 2004, http://www.javvin.com/protocolIPv6.html

[Kam04] KAME Project, 2004, http://www.kame.net

[KeA98a] S. Kent and R. Atkinson, “RFC 2402 – IP Authentication Header,” November 1998, http://www.ietf.org/rfc/rfc2402.txt

[KeA98b] S. Kent and R. Atkinson, “RFC 2406 – IP Encapsulating Security Payload,” November 1998, http://www.ietf.org/rfc/rfc2406.txt

[KeA98c] S. Kent and R. Atkinson, “RFC 2401 – Security Architecture for the Internet Protocol,” November 1998, http://www.ietf.org/rfc/rfc2401.txt

[Kra04] G. Kramer, "On generating self-similar traffic using pseudo-Pareto distribution," Technical brief, Department of Computer Science, University of California, Davis, 2004, http://wwwcsif.cs.ucdavis.edu/~kramer/papers/self_sim.pdf

[Mca04] McAfee Security, “Encrypted Threat Protection: Network IPS for SSL Encrypted Traffic,” June 2004, http://www.mcafeesecurity.com/us/_tier2/products/_media/mcafee/wp_encr_th_prot.pdf

[Mic03] Microsoft Corporation, "Frequently Asked Questions about the IPv6 Protocol for Windows XP," May 2003, http://www.microsoft.com/technet/prodtechnol/winxppro/plan/faqipv6.mspx

[Mic04] Microsoft Security Website, 2004, http://www.microsoft.com/security

[Nis04] National Institute of Standards and Technology, “Advanced Encryption Standard (AES), Data Encryption Standard (DES), Triple-DES, and Skipjack Algorithms,” 2004, http://csrc.nist.gov/cryptval/des.htm

[Nss04] The NSS Group, “Gigabit Intrusion Detection Systems,” January 2004, http://www.nss.co.uk/WhitePapers/gigabit_ids.htm

[Pca04] The libpcap Project, 2004, http://sourceforge.net/projects/libpcap/

[Ran02] M. Ranum, “Intrusion Detection: Challenges and Myths,” October 2002, http://secinf.net/info/ids/ids_mythe.html

[Roe04] M. Roesch et al, *Snort – The Open Source NIDS*, 2004, http://www.snort.org

[Saf04] SafeNet SafeXcel I80-PCI Card Data Sheet, 2004, http://www.safenet-inc.com/

[Sch04] Michael Schorr, "IPv4 and IPv6: A Comparison," January 2004, http://myitforum.techtarget.com/articles/16/view.asp?id=6720

[Shi02] R. Shimonski, "What You Need to Know About Intrusion Detection Systems," November 2002, http://www.windowsecurity.com/articles/What_You_Need_to_Know_About_Intrusion_Detection_Systems.html

[Shi03] D. Shindler, "IPv6: Windows Server 2003 Supports a More Secure IP – Sort of," November 2003, http://www.windowsecurity.com/articles/Windows_Server_2003_IPv6.html

[Sou03] SourceForge.net Snort Mail Archives, Conversation with Nigel Hougton, 2003, http://sourceforge.net/mailarchive/forum.php?forum_id=3972&style=flat&viewday=22&viewmonth=200310

[Sun04] Sun Crypto Accelerator 4000 Board Data Sheet, 2004, http://www.sun.com/networking

[Tay02] L. Taylor, "Lock IT Down: Intrusion detection is not intrusion prevention," August 2002 http://techrepublic.com.com/5100-6264-1051215.html

[Tri03] A. Triulzi, "Intrusion Detection in IPv6," Velikonoční kryptologie 2003

[Tro04] Tropical Software, "DES Encryption," 2004, http://www.tropsoft.com/strongenc/des.htm

[War03] M. Warfield, "Security Implications of IPv6," 2003, http://documents.iss.net/whitepapers/IPv6.pdf

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)* 21-03-2005	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* September 2003 – March 2005

4. TITLE AND SUBTITLE

ENABLING INTRUSION DETECTION IN IPSEC PROTECTED IPV6 NETWORKS THROUGH SECRET-KEY SHARING

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Sweeney, Patrick J., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/05-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Security Agency
Attn: Mr. Ken Shotting, I44
9800 Savage Road
Fort Meade, MD 20755-6744
(410) 854-6761

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

As the Internet Protocol version 6 (IPv6) implementation becomes more widespread, the IP Security (IPSec) features embedded into the next-generation protocol will become more accessible than ever. Though the network-layer encryption provided by IPSec is a boon to data security, its use renders standard network intrusion detection systems (NIDS) useless. The problem of performing intrusion detection on encrypted traffic has been addressed by differing means with each technique requiring one or more static secret keys to be shared with the NIDS beforehand. The problem with this approach is static keying is much less secure than dynamic key generation through the Internet Key Exchange (IKE) protocol.

This research creates and evaluates a secret-key sharing framework which allows both the added security of dynamic IPSec key generation through IKE, and intrusion detection capability for a NIDS on the network. Analysis shows that network traffic related to secret-key sharing with the proposed framework can account for up to 58.6% of total traffic in the worst case scenario, though workloads which are arguably more average decrease that traffic to 10-15%. Additionally, actions associated with IKE and secret-key sharing increase CPU utilization on the NIDS up to 20.7%. Results show, at least in limited implementations, a secret-key sharing framework provides robust coverage and is a viable intrusion detection option.

15. SUBJECT TERMS

Intrusion detection, communications protocols, computer security, secure communications, IPv6

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Richard A. Raines, AFIT/ENG
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4278 (Richard.Raines@afit.edu)
U	U	U	UU	95	